(19)日本国特許庁（ＪＰ）　(12)公表特許公報（Ａ）　(11)特許出願公表番号

特表平9－506191

(43)公表日　平成９年(1997)６月17日

(51)Int.Cl.[6]		識別記号	庁内整理番号	ＦＩ		
Ｇ０６Ｆ	3/14	３４０	9174－5E	Ｇ０６Ｆ	3/14	３４０Ａ
		３１０	9174－5E			３１０Ｃ
	9/06	５３０	9367－5B		9/06	５３０Ｎ
Ｇ０６Ｔ	11/80		9365－5H		15/62	３２２Ｂ
			9365－5H			３２０Ｋ

審査請求　未請求　予備審査請求　有　（全 53 頁）

(21)出願番号　特願平7－512576
(86)(22)出願日　平成６年(1994)１月３日
(85)翻訳文提出日　平成８年(1996)４月30日
(86)国際出願番号　ＰＣＴ／ＵＳ９４／００００５４
(87)国際公開番号　ＷＯ９５／１２１６１
(87)国際公開日　平成７年(1995)５月４日
(31)優先権主張番号　０８／１４６，６３１
(32)優先日　1993年10月29日
(33)優先権主張国　米国（ＵＳ）

(71)出願人　オブジェクト　テクノロジー　ライセンシング　コーポレイション
アメリカ合衆国　95014　カリフォルニア州　クパチーノ　ノース　デ　アンザ　ブールバード　10355
(72)発明者　シードル，ロバート
アメリカ合衆国　94303　カリフォルニア州　パロ　アルト　コロニアル　レーン　946
(74)代理人　弁理士　谷　義一　（外１名）

最終頁に続く

(54)【発明の名称】　グラフィック・エディタ・フレームワーク・システム

(57)【要約】
グラフィック・エディタ・フレームワーク・システムが開示されている。具体的には、アプリケーション相互間のグラフィカル・データのやりとりを取り扱い、グラフィカル・オブジェクトを表示し、操作するためのフレームワークを含めてグラフィックス・アプリケーションを開発するための方法と装置が開示されている。フレームワークは、アプリケーション開発者によって使用されてシステム・アーキテクチャの主要サブシステムであるモデル、ビューおよびユーザ・インタフェース間のやりとりを容易化する多数のクラスを含んでいる。

【特許請求の範囲】

1．接続されたディスプレイを装備したコンピュータ上に置かれたオブジェクト指向オペレーティング・システムと一緒に使用され、対話式グラフィカル・ユーザ・インタフェースを構築するためのオブジェクト指向フレームワークであって、

（a）グラフィカル・ユーザ・インタフェースを構成するグラフィカル・オブジェクトをモデル化する手段と、

（b）ディスプレイ上でグラフィカル・オブジェクトをレンダリングする手段と、

（c）ディスプレイ上のグラフィカル・オブジェクトをセレクション・オブジェクトを使用して選択する手段と、

（d）ディスプレイ上のグラフィカル・オブジェクトをコマンド・フレームワークを使用して変更する手段と

を備えていることを特徴とするオブジェクト指向フレームワーク。

2．スクリーン更新メカニズムを選択的にイネーブルする手段を含むことを特徴とする請求項1に記載のオブジェクト指向フレームワーク。

3．ディスプレイに直接にレンダリングするアップデータ・オブジェクトを含むことを特徴とする請求項2に記載のオブジェクト指向フレームワーク。

4．単一バッファ付きアップデータ・オブジェクトを含むことを特徴とする請求項2に記載の方法。

5．2重バッファ付きアップデータ・オブジェクトを含むことを特徴とする請求項2に記載の方法。

6．3重バッファ付きアップデータ・オブジェクトを含むことを特徴とする請求項2に記載の方法。

7．複数のconstraint（制約）オブジェクトによって制約されるカーソルを含むことを特徴とする請求項1に記載の方法。

8．位置によって制約するconstraintオブジェクトを含むことを特徴とする請求項1に記載の方法。

９．幾何学的プロパティを利用するconstraintオブジェクトを含むことを特徴とする請求項１に記載の方法。

１０．幾何学的プロパティは矩形グリッドを含むことを特徴とする請求項９に記載の方法。

１１．幾何学的プロパティはポーラ（極）・グリッドであることを特徴とする請求項９に記載の方法。

１２．幾何学的プロパティは遠近グリッドであることを特徴とする請求項９に記載の方法。

１３．セマンティック制約を利用するconstraintオブジェクトを含むことを特徴とする請求項８に記載の方法。

１４．constraintオブジェクトはコーナ、サイド、中心、中間点などの、オブジェクト上の有意点にスナップすることを特徴とする請求項１３に記載の方法。

１５．constraintオブジェクトはグラフィカル・オブジェクト上のコネクション・ポートにスナップすることを特徴とする請求項１３に記載の方法。

１６．プログラムが動作状態にあるときconstraintオブジェクトを動的にやりとりするステップを含むことを特徴とする請求項７に記載の方法。

１７．グラフィカル・オブジェクトの代わりにセレクション・フィードバッカ・オブジェクトを表示し、マウス・ダウン・イベントといったイベントをグラフィカル・オブジェクトの代わりにセレクション・フィードバッカに渡す機能を備えたセレクション・フィードバッカ・オブジェクトを含むことを特徴とする請求項１に記載の方法。

１８．セレクション・フィードバッカ・オブジェクトは、プログラムが動作状態にあるとき相互のために動的にインスタンス生成またはやりとり可能であることを特徴とする請求項１７に記載の方法。

１９．方向によって制約するconstraintオブジェクトを含むことを特徴とする請求項１に記載の方法。

【発明の詳細な説明】

グラフィック・エディタ・フレームワーク・システム

発明の背景

発明の分野

本発明は一般的には、モデル、ビュー・システムおよびユーザ・インタフェース間の相互作用（インタラクション－相互にやりとりすること）を単純化するオブジェクト指向フレームワークに関する。本発明は、ポピュラーなオブジェクト指向プログラミング言語であるC++を使用した好適実施例を中心にして説明されているが、その原理はオブジェクト指向と手続き型の両方の、他のコンピュータ・プログラミング言語にも応用可能である。

従来技術の説明

オブジェクト指向プログラミング(Object-oriented programming - OOP)は、ユーザに親しみやすく（ユーザ・フレンドリ）、インテリジェントなコンピュータ・ソフトウェアを構築するのに好適な環境である。ＯＯＰのキー・エレメントはカプセル化(encapsulation)、継承(inheritance)および多態(polymorphisim)である。これらのエレメントは、アイコン、マウス・カーソルおよびメニューをもつウィンドウ操作環境で代表される特徴をもつ、グラフィカル・ユーザ・インタフェース(graphical user interface - GUI)を生成するために使用されている。これらの３つのキー・エレメントはＯＯＰ言語に共通しているが、大部分のＯＯＰ言語はこれらの３つのキー・エレメントの実現方法が異なっている。

ＯＯＰ言語の例としては、SmallTalk、Object Pascal およびC++がある。Smalltalk は実際には言語以上のものであり、もっと正確にいうと、プログラミング環境と特徴づけることができる。Smalltalk は、１９７０年代の初めにゼロックス社 Palo Alto Research Center(PARC)のLearning Research Group によって開発された。Smalltalk では、メッセージがオブジェクトへ送られて、そのオブジェクト自体を評価している。メッセージは、従来のプログラミング言語におけるファンクション・コール（関数呼出し）のそれと似たタスクを実行する。プログラマはデータのタイプ（型）を気にする必要がなく、むしろ、プログラマは

メッセージを正しい順序で作成し、正しいメッセージを使用することだけに気をくばればよい。Object Pascal はアップル社マッキントッシュ（登録商標）コンピュータ用の言語である。アップル社は、Pascalの設計者であるNiklaus Wirthの協力を得てObject Pascal を開発した。Ｃ++は１９８３年に、Ｃの拡張版としてAT&T Bell LaboratoriesのBjarne Stroustrup によって開発された。Ｃ++の中心となる概念はクラス(class)であり、これはユーザが定義するタイプである。クラスはオブジェクト指向プログラミングの機能（特徴）を備えている。Ｃ++のモジュールはＣのモジュールと互換性があるので、既存のＣライブラリをＣ++プログラムで使用できるように自由にリンクすることができる。最も普及しているオブジェクト・ベースおよびオブジェクト指向のプログラミング言語の起源は、１９６０年代にノルウェーのO-J. Dahl、B. MyhrhaugおよびK. Nygrad によって開発されたSimulaにさかのぼる。ＯＯＰの主題に関する詳細情報は、Grady Booch 著「オブジェクト指向設計とその応用(Object-oriented designs with Applications)」 (Benjamin/Cummings Publishing Co., Inc., Redwood City, Calif.(1991)に記載されている。

発明の概要

従って、本発明の目的は、グラフィックス・アプリケーションを構築するためのオブジェクト指向フレームワークを提供することである。このフレームワークは、グラフィック情報をオペレーティング・システムのサブシステム相互間とアプリケーション内でやりとりすることを容易化するための、種々の関数を実現する多数のクラスを含んでいる。さらに、フレームワークは、より複雑な機能や関数を必要とする場合には、アプリケーション設計者がカストマイズしたり、オーバライドしたりできるようになっている。

本発明によれば、グラフィカル編集機能を使用してアプリケーションを開発するためのオブジェクト指向フレームワークが用意されており、このフレームワークには、システム・アーキテクチャのサブシステム相互間およびあるアプリケーションと他のアプリケーションとの間のデフォルト（省略時）のやりとりを定義するためのクラスが多数用意されている。クラスは、グラフィック・オブジェク

トとデータをドローイング（描画）し、やりとりし、操作し、表示するためのメソッドを用意している。

図面の簡単な説明

上記および他の目的、側面および利点の理解を容易にするために、以下では、添付図面を参照して本発明の好適実施例について説明する。図面において、

図１は好適実施例によるコンピュータ・システムを示すブロック図である。

図２は、好適実施例においてグラフィカル・エディタ・フレームワークの基本的基底クラスが相互に作用し合う様子(interaction：やりとり)を示す図である。

図３Ａは、好適実施例においてあるコンポーネントの基底クラスと他のクラスとの関係を示す図である。

図３Ｂは、好適実施例によるグラフィック・モデルにおけるコンポーネントを示す図である。

図４Ａ、図４Ｂ、図４Ｃおよび図４Ｄは、好適実施例による種々の更新手法を示す図である。

図５は、好適実施例によるディスプレイ・スクリーン上の９個のピクセルを拡大して示す図である。

図６は好適実施例によるスナップツー(snap-to)・オブジェクトを示す図である。

図７は、好適実施例によるセマンティック・スナッピング・オペレーションが

完了したことを示す図である。

図８は、好適実施例による矩形グリッド８００と等角(isometric)グリッド８１０を示す図である。

図９は、好適実施例においてアーキテクチャに基づくレンダリングの例を示す図である。

図１０は好適実施例による遠近グリッドを示す図である。

図１１は、好適実施例において２点遠近グリッドを使用して作られた立方体を示す斜視図である。

図１２は、好適実施例によるロケーション制約フレームワークに関連する詳細ロジックのフローチャートを示す図である。

図１３は、好適実施例によるアイドル・スナップ・フェーズに関連する詳細ロジックのフローチャートを示す図である。

図１４は、好適実施例によるトラック開始フェーズに関連する詳細ロジックのフローチャートを示す図である。

図１５は、好適実施例によるトラック継続フェーズに関連する詳細ロジックのフローチャートを示す図である。

図１６は、好適実施例によるトラック終了フェーズに関連する詳細ロジックのフローチャートを示す図である。

発明の好適実施例の詳細な説明

本発明は、好ましくは、ＩＢＭ（登録商標）ＰＳ／２（登録商標）またはアップル社（登録商標）マッキントッシュ（登録商標）コンピュータなどのパーソナル・コンピュータに置かれたオペレーティング・システムを背景に実施される。図１は代表的なハードウェア環境を示しもので、本発明によるワークステーションの代表的なハードウェア構成を示していてる。このハードウェア構成は、従来のマイクロプロセッサなどの中央処理ユニット１０、およびシステム・バス１２を介して相互接続された複数の他のユニットを含んでいる。図１に示すワークステーションはランダム・アクセス・メモリ（ＲＡＭ）１４、リードオンリ・メモリ（ＲＯＭ）１６、ディスク・ユニット２０などの周辺デバイスをバスに接続するための入出力アダプタ１８、キーボード２４、マウス２６、スピーカ２８、マイクロホン３２および／またはタッチスクリーン・デバイス（図示せず）などの他のユーザ・インタフェース・デバイスをバスに接続するためのユーザ・インタフェース・アダプタ２２、ワークステーションをデータ処理ネットワークに接続するための通信アダプタ３４、およびバスをディスプレイ・デバイス３８に接続するためのディスプレイ・アダプタ３６を装備している。ワークステーションには、アップル・システム／７（登録商標）オペレーティング・システムなどのオペレーティング・システムが常駐している。

好適実施例において、本発明はオブジェクト指向プログラミング技法を用いてC++プログラミング言語で実現されている。この分野の精通者ならば理解されるように、オブジェクト指向プログラミング(Object-Oriented Programming - OOP)のオブジェクトは、データ構造と、データに対するオペレーション（操作、演算など）とからなるソフトウェア・エンティティである。これらのエレメントが一体になって、オブジェクトは、そのデータ・エレメントで表されたその特性と、そのデータ操作関数で表されたその振舞い(behavior－作用ともいう)とからとらえて、ほとんどどのような実世界のエンティティでもモデル化することが可能である。このようにすると、オブジェクトは人やコンピュータのような具体物をモデル化することも、数や幾何学的概念のような抽象的概念をモデル化することもできる。オブジェクト・テクノロジの利点は３つの基本原理、つまり、カプセル化(encapsulation)、多態(polymorphism)および継承(inheritance)に起因するものである。

オブジェクトは、そのデータの内部構造および関数が作用するときのアルゴリズムを外部から見えないように隠している。つまり、カプセル化している。これらのインプリメンテーション（実現、実装方法）の詳細を見せる代わりに、オブジェクトは、外部の情報がないクリーンな形で、抽象化を表したインタフェースを提示している。多態はカプセル化を一歩進めたものである。その考え方は多数の形をもつ、１つのインタフェースである。ソフトウェア・コンポーネントは、別のコンポートネントがなんであるかを正確に知らなくても、そのコンポーネントの要求を行うことができる。要求を受け取ったコンポーネントはその要求の意味を理解し、要求をどのように実行すべきかを、その変数とデータに従って判断する。第３の原理である継承によれば、開発者は既存の設計とデータを再利用することができる。この機能によると、開発者はソフトウェアを初めから作る必要がなくなる。むしろ、継承によれば、開発者は振舞いを継承するサブクラスを派生し、その振舞いを具体的要求に合わせてカストマイズすることになる。

従来の解決手法（アプローチ）では、オブジェクトとクラス・ライブラリを手続き型環境(procedural environment)で階層化している。市販されている多くの

アプリケーション・フレームワークはこの設計手法を採用している。この設計では、１つまたは２つ以上のオブジェクト層（レイヤ）がモノリシック・オペレーティング・システム(monolithic operating system)の上に上乗せされている。この手法では、カプセル化、多態および継承の原理をすべてオブジェクト層に採用し、手続き型プログラミング技法を大幅に改善しているが、この設計にはいくつかの制約がある。これらの困難性は、開発者が開発者自身のオブジェクトを使用するのは容易であるが、他のシステムからのオブジェクトを使用することが困難であること、および、依然として手続き型オペレーティング・システム（ＯＳ）のコールを使用して下位の非オブジェクト層まで到達する必要があることに起因している。

オブジェクト指向プログラミングがもつもう１つの側面は、アプリケーション開発にフレームワーク手法を採用していることである。フレームワークの最も合理的な定義の１つとして、University of IllinoisのRalph E. JohonsonおよびPurdueのVincent F. Russoによるものがある。１９９１年の論文「オブジェクび指向設計の再利用(Reusing Object-Oriented Designs)」（University of Illinoisテクニカル・レポートUTUCDCS91-1696）において、次のような定義が提案されている。「抽象クラスとは、共同に働いて１組の責任分担を遂行する１組のオブジェクトを設計したものである。従って、フレームワークとは、共同作用して定義された組の計算責任分担を実行する１組のオブジェクト・クラスである。」プログミング側から見たとき、フレームワークは、基本的には、稼働するアプリケーションのあらかじめ作られた構造を提供する、相互に接続されたオブ

ジェクト・クラスの集まりである。例えば、ユーザ・インタフェース・フレームワークは、ドローイング・ウィンドウ、スクロールバー、メニューなどのサポートを行い、デフォルトの振舞いを用意している。フレームワークはオブジェクト・テクノロジを基礎にしているので、この振舞い(behavior)を継承し、オーバライドすることにより、開発者はフレームワームを拡張し、カストマイズした問題解決手法を特定の専門知識分野で作ることができる。これが従来のプログラミングに比べて大きな利点となっているのは、プログラマはオリジナル・コードを変

更するのではなく、ソフトウェアを拡張するからである。さらに、開発者がコード層の初めから最後に至るまで盲目的に作業していないのは、フレームワークには構造的に関するガイダンスとモデリングが用意されているが、それと同時に、開発者は問題分野に固有の特定のアクションを用意することから解放されるからである。

ビジネス側から見たときは、フレームワークは特定の知識分野における専門知識をカプセル化または具現化する方法と見ることができる。企業の開発集団、独立ソフトウェア・ベンダ(Independent Software Vendors - ＩＳＶ)およびシステム・インテグレータは、前述した例におけるように、製造、会計、または通貨取引きなどの特定分野における専門知識を習得している。この専門知識はコードで具現化されている。フレームワークによると、企業はその専門知識の共通特性を収集したあとで、専門知識を企業のコードで具現化することにより専門知識の共通特性をパッケージ化することができる。この結果、第１に、開発者は専門知識を利用するアプリケーションを作成あるいは拡張することができ、問題がいったん解決されると、ビジネル・ルールと設計が実施され、統一的に使用されることになる。さらに、フレームワークと、フレームワークの背後にある具現化された専門知識は、製造、会計、バイオテクノロジなどの垂直マーケットにおいて専門知識を習得した企業にとっては、戦略的資産の意味合いをもつことになり、その企業は、専門知識をパッケージ化し、再販し、分散化し、さらには、テクノロジの進歩と普及化を促進するための流通メカニズムをもつことになる。

歴史的には、フレームワークがパーソナル・コンピューティング・プラットフォームで主流概念として出現したのはつい最近のことである。この移行を助長したのは、C++などのオブジェクト指向言語が利用可能になったことである。伝統的に、C++は、商用化されたパーソナル・コンピュータにではなく、大部分がＵＮＩＸシステムや研究者のワークステーションに見られていた。いくつかの大学や研究所のプロジェクトが今日の商用化フレームワークとクラス・ライブラリの先駆けとなることを可能にしたのは、C++などの言語とSmalltalk その他などの、他のオブジェクト指向言語である。これらの例のいくつかを挙げると、Stan

ford University のInterViews、Carnegie-Mellon UniversityのAndrew toolkit、およびUniverstity of Zurich のET++フレームワークがある。

フレームワークには、どのレベルのシステムに関心があるか、どのような問題を解決しようとしているかに応じて、多数の種類のものがある。フレームワークのタイプは、ユーザ・インタフェースの開発を支援するアプリケーション・フレームワークから、通信、印刷、ファイル・システム・サポート、グラフィックスなどの基本的システム・ソフトウェア・サービスを提供する低レベル・フレームワークまでの範囲にわたっている。商用化されているアプリケーション・フレームワーク例をいくつか挙げると、MacApp(Apple社)、Bedrock(Symantec社）、OWL(Borland社)、NeXTStep App Kit (NeXT社) およびSmalltalk-80MVC (ParcPlace社) がある。

フレームワークを使用したプログラミングを行うためには、他の種類のシステムに使い慣れた開発者は考え方を変える必要がある。事実、これは、従来の「プログラミング」とは全く異なっている。ＤＯＳやＵＮＩＸなどの旧スタイルのオペレーティング・システムでは、開発者自身のプログラムが構造のすべてを提供している。オペレーティング・システムはシステム・コールを通してサービスを提供している。つまり、開発者のプログラムはサービスが必要になったとき、コールを行い、サービスが提供されるとコントロールを返却している。プログラム構造はコントロールの流れに基づいており、これは開発者が書いたコードで実現されている。

フレームワークが使用されるときは、上記とは反対である。開発者はコントロールの流れに責任を持つ必要がなくなった。開発者はプログラミング・タスク

を実行の流れからとらえて理解する傾向をなくす必要がある。むしろ、オブジェクトの責任分担からとらえた考え方をして、タスクをいつ実行させるかの判断はフレームワークに任せる必要がある。開発者が書いたルーチンはその開発者以外の人が書いた、しかも、その開発者が見たこともないコードによってアクチベートされる。このコントロールの流れのフリップフロップは、手続き型プログラミングの経験しかない開発者にとっては、大きな心理的障害となっている。しかし

、このことを理解してしまえば、フレームワーク・プログラミングは、他のタイプのプログラミングよりも作業量が大幅に軽減される。

アプリケーション・フレームワークがプレハブ機能を開発者に提供するのと同じように、好適実施例で取り入れられているようなシステム・フレームワークもこの考え方をてこにして、システム・レベルのサービスを提供しているので、システム・プログラマなどの開発者は、これらのサービスをサブクラス化するために使用するか、オーバライドすることにより、カストマイズした問題解決手法を作成することができる。例えば、オーディオ、ビデオ、ＭＩＤＩ、アニメーションなどの新しく多様なデバイスをサポートする土台を提供できるマルチメディア・フレームワークを考えてみる。新しい種類のデバイスをサポートする必要が起こったとき、開発者はデバイス・ドライバを書く必要があった。これをフレームワークを使用して行うと、開発者は、その新デバイスに特有の特性と振舞い(behavior)を供給するだけで済むことになる。

この場合には、開発者は、マルチメディア・フレームワークによってコールされるある種のメンバ関数のインプリメンテーションを用意する。開発者にとって直接の利点は、デバイスの各カテゴリ別に必要になる汎用コード(generic code)がすでにマルチメディア・フレームワークに用意されていることである。このことは、デバイス・ドライバの開発者が書き、テストし、デバッグするコードが少なくなることを意味する。システム・フレームワークを使用する別の例として、入出力フレームワークがＳＣＳＩデバイス、ＮｕＢｕｓカード、およびグラフィックス・デバイス別に用意されることである。継承された機能があるので、各フレームワークは、そのデバイス・カテゴリに見られる共通機能に対するサポートを用意している。そうすれば、他の開発者は、これらの統一インタフェース

スを通してあらゆる種類のデバイスと結ぶことが可能になる。

好適実施例では、フレームワークの考え方を取り入れ、システム全体に適用している。取引先または企業の開発者、システム・インテグレータ、またはＯＥＭにとっては、このことは、MacAppのようなフレームワークについて説明してきたすべての利点が、テキストやユーザ・インタフェースなどの事物についてはアプ

リケーション・レベルで、グラフィックス、マルチメディア、ファイル・システム、入出力、テストなどのサービスについてはシステム・レベルで生かすことができることを意味する。好適実施例のアーキテクチャでのアプリケーション作成は、基本的には、フレームワーク・プロトコルに準拠する問題領域特有のパズルピースを書くのと似ている。このように、プログラミングの考え方全体が変わっている。複数のＡＰＩ階層をコールするコードを１行ずつ書くのではなく、ソフトウェアは、この環境内の既存フレームワークからクラスを派生し、そのあとで、必要に応じて新しい振舞いを追加し、および／または継承した振舞いをオーバライドすることによって開発される。

以上のように、開発者のアプリケーションは、書かれたあと、他のすべてのフレームワーク・アプリケーションと共有されるコードのコレクション（集まり）となる。これは、開発者がお互いの作業に積み上げていくことができる点で強力な概念となっている。また、開発者は、必要な量だけカストマイズできるという柔軟性も得られる。フレームワークによっては、そのままで使用されるものもある。ある場合には、カストマイズ量が最小になるので、開発者がプラグインするパズルピースは小さくなる。他の場合には、開発者は大幅な変更を行って、まったく新しいものを作成することもある。本発明を理解する上で重要なことは、「フレームワーク」の概念を理解し、フレームワークと「オブジェクト」および「オブジェクト指向プログラミング」との関係を理解することである。フレームワークとそこに具現化されている基本概念を説明した初期の論文として、Kurl A. Schmucker 著「MacApp: アプリケーション・フレームワーク」（１９８６年８月号のByte誌に掲載）があるが、この論文の全文は引用により本明細書の一部を構成するものである。オブジェクトの重要な特徴は、オブジェクトが受け持つデータとメソッドをカプセル化できることである。つまり、あるオ

ブジェクトがコマンドをどのように実行するかの内部詳細を他のオブジェクトが知らなくても、汎用コマンドをオブジェクトに出すことができる。同じ意味において、コマンド、データ、ファイル名などにグローバルな互換性をもたせる必要がないので、オブジェクト同士を自由に関連づけることができる。フレームワー

クは、本質的には、オブジェクトのクラスを関連づけたものからなる汎用アプリケーションであるので、必要に応じて他のオブジェクトと関連づけることにより、より特殊化されたアプリケーションを作ることができる。オブジェクトのクラスを関連づけたものとしてのフレームワークでは、オブジェクトのクラス相互間の機能上の関係がそこに定義されているため、フレームワークと関連づけることができる追加オブジェクトの汎用または特定機能を、必要とするどのレベルでも得ることができる。

従って、フレームワークは、オブジェクト間の暗黙の責任分担ネットワークを提供し、オブジェクトのクラス間の継承を可能にし（例えば、オブジェクトのクラスの上位階層レベルに置かれたスーパクラスのデータとメソッド）、およびイベントに応じてライブラリのコールが行えるようにするシステムと見ることができる。フレームワークとして構築されたシステムは、より特殊化した関数を実行するオブジェクトや、フレームワークに用意されている関数をオーバライドすることもできるオブジェクトを追加することにより、カストマイズすることも可能である。フレームワークの種々のクラスとサブクラスにおけるマシン特有およびデバイス特有のオブジェクトを使用すると、フレームワーク自体をマシン独立およびデバイス独立にして、応用に汎用性をもたせることができる。さらに、特定のフレームワークを特徴づけているのは、オブジェクトとオブジェクトのクラス相互間の関係が責任の分担と、その結果として達成される継承と機能性からとらえて確立されていることである。また、フレームワーク自体を、特定のアプリケーションを開発するときのテンプレートとして使用すれば、カストマイズと機能上のオーバライドを特定のオブジェクトとしてそのアプリケーションに用意することも可能である。

グラフィカル・エディタ・モデルはオブジェクト指向プログラミングの原理に基づいている。オブジェクト指向プログラミングの一般的概念は簡単に上述した

が、これらは公知であるので、ここで詳しく説明することは省略する。概要を説明すると、データは抽象化され、カプセル化されており、グラフィック・オブジェクト情報を表している、あるいはその情報を収めているオブジェクトは、アー

キテクチャ全体を変えることなく、可変データ・フォーマットで表されている。オブジェクトとのインタフェースは一定のままであり、オブジェクト自体は抽象化され、相互に独立している。

オブジェクト指向プログラミング設計におけるクラスまたはオブジェクトは、構造（例えば、データ）とその構造に作用する振舞い（例えば、「メソッド関数」と呼ばれているもの）をカプセル化している。オブジェクト指向の設計では、インタフェースはクラスまたはオブジェクトを外から見たものであり、クラスまたはオブジェクトの構造と振舞いは外部から見えないようにしている。さらに、基底クラスから派生するすべてのオブジェクトは基底クラスのプロパティを継承しているので、基底クラスと同じプロパティをもつことになり、基底クラスのオペレーションに対して多態化されている。従って、基底クラスから派生するオブジェクトは基底クラスのインスタンスを表すために使用することができ、基底クラスがコールされるときは、いつでも、これを代用することができる。

グラフィカル・エディタ・フレームワーク(GrafEdit)には、基本的基底クラスとしてModel（モデル）、Component（コンポーネント）、Canvas（カンバス）の３つがある。図３Ａは、これら３つの基本的基底クラス間の関係を示す概要図である。図３Ａに示すように、Model は Componentを収容するのに対し、Canvasクラスは情報を表示し、Component のグラフィカル・オブジェクト・タイプに対する変更を反映している。

Model 基底クラスであるTGrafEditModelはデータ・ストア・クラスであり、複数のグラフィック・コンポーネントと他のモデル（GrafEditとそうでない場合の両方）を収容することができる。Model クラスはComponent メソッドへの基本アクセスを定義しているが、メソッドのストレージ・インプリメンテーションは定義していない。さらに、Model クラスはルート（根）Model として使用することができる。このクラスは、定義されたメソッドがコンポーネントと、従ってそのデータをアクセスするための、コンポーネントのストアの働きをする。デフォルトのインプリメンテーションでは、単純なフラット・ストレージ・システムである順序付リスト(ordered list)を使用して、コンポーネントをストアしている。

異なるストレージ・システムが必要であるときは、開発者は、TGafEditModel からサブクラス化するという方法でこのシステムをオーバライドしてカストマイズすることができる。Model 基底クラスには、Model 内のコンポーネントに対してアクションを実行するメソッドがいくつか用意されている。

第１のメソッド群によると、Component データの追加、削除およびアクセスを行うことができる。別のメソッドとして、コンポーネントの順序を変更できるようにするものが用意されている。これが重要であるのは、デフォルトとして、情報がスクリーン上にドローイングするために使用されるからである。Component データを走査するイタレータ(iterator)を作成するメソッドも用意されている。イタレーション（繰返し）には、順序付(ordered)と、前面から後面へ(front to back)または順序なし(unordered)の２種類がある。Model 基底クラスには、Model 全体にわたるメニューを指定し、追加できるようにするメソッドも用意されている。最後に、フレームワークによって使用されるGrafEdit Selectionのすべてを作成するためのメソッドが用意されている。これにより、アプリケーション開発者はGrafEdit Selectionクラスをオーバライドすることができる。コンポーネントに影響するアクションを実行するために用意されたメソッドは、２通りの方法で実行することができる。最初は直接コールによる方法である。もう１つはModelCommandを使用してメソッドをコールする方法であり、このメカニズムによると、コマンドを取り消す(undo)ことが可能である。

Component 基底クラスであるTGraphicComponent は、グラフィック・コンポーネントをストアし、ドローイング、相互作用(interaction)および操作メソッドを定義している。図４は、TGraphiComponentと、グラフィック・オブジェクトを取り扱う他のクラスとの基本的関係を示している。同図に示すように、これらのクラスはすべてがMGraphicに従属しており、このことは、MGraphicの機能を継承していることを意味する。上述したように、TGraphicComponent はコンポーネントの抽象基底クラス(abstruct base class)である。図示の残りのクラスはインプリメンテーションを示しており、TSimplestComponentは、ラップされた

MGraphicを用いてTGraphicComponent メソッドのすべてをインプリメントし、TL

ineComponentは具象クラス(concrete class)の例である。

MGraphicの継承された機能のほかに、TGraphicComponent には、GrafEditセレクションにストアされ、コンポーネントを比較するとき使用されるコンポーネントのユニークな識別子が用意されている。また、Component を編集するためのインタラクタ(interactor)、Component の振舞いを指定するためのセレクション・フィードバッカ(selection feedbacker)、およびコンポーネント・ソケット全体にわたって繰り返すイタレータ(iterator)も用意されている。

具象インプリメンテーションの例をいくつか示すと、TSimplestComponentとTLineComponentがある。TSimplestComponentは、MGraphicを使用してTGraphicComponent 仮想（バーチャル）メソッドを実装（インプリメント）するインプリメンテーションである。このインプリメンテーションは任意のMGraphics からコンポーネントを多態的に作成するときに使用すると便利であるが、これが行われるのは、非GrafEdit「アプリケーション」がそのMGraphics をコンポーネントに変換したいときである。これに対して、TLineComponentは、階層をどこでサブクラス化するかを示すために使用される。

TCanvasView

TCanvasView はTView のサブクラスである。これが、GrafEditによって使用される主要ルーチンの１つとなっているのは、ドキュメントを準備して表示し、ユーザがドキュメントとやりとりするのを容易にするからである。ドキュメント・スタートアップ・シーケンスでモデルのCreateEditablePresentationがコールされると、これはカンバス・ビュー(canvas view)を作成する。例えば、メニュー・アイテムは、独立にスクロールできる別のビューを作成することができる。モデル内のデータが変更されると、すべてのビューに通知される。基本的には、同じデータのプレゼンテーションはビューごとに異なっている。そのため、あるコンポーネントが一方のビューでドラッグされると、そのコンポーネントは他方のビューで自動的に移動する。これが共同作用(collaboration)と

同じでないのは共同作用は、アドレス空間(address space)を横断して行われるが、その効果が同じように見えるからである。コンポーネント・カンバスTCompo

nentCanvasのコンストラクタ(constructor)は、通常のサイズとロケーションのほかにモデルとモデル引数を受け取る。オプションの引数であるfitToView もあるが、これは表示されるグラフィック・マテリアルの内容とは無関係である。その代わりにTRUEにセットされていれば、カンバスは、すべての内容が常に表示できるようにそのプレゼンテーションをスケーリングする。この機能は、Scrapbook で使用されているもののように、カンバスを寸描表示する場合に便利である。

更新ポリシー

オンスクリーン・イメージを更新する方法は多数ある。どの更新プロシージャも、更新品質（フリッカリング）とメモリ・コスト（バッファ）と更新スピード（再ドローイングとコピービットとの対比）との妥協的産物である。更新ポリシーの詳細をカンバス・インプリメンテーションの残り部分から隔離すると、コンポーネント・カンバスをサブクラス化しなくても、やりとり(exchange)や再インプリメンテーションを容易化することができる。コンポーネント・カンバスはTCanvasUpdaterのサブクラスであるオブジェクトをもっているので、ポリシーの意思決定が容易化されている。カンバス・クラスは、プラグ可能なアップデータ(updater)をサポートするデータ構造も維持している。実際には、モデル内のすべてのコンポーネントをイクイバレンス・クラス(equivalence class)に分割している３つのリストを維持している。１．バックグラウンド・オブジェクト：全体が現在選択されているすべてのオブジェクトの背後にあるすべてのオブジェクト。２．フォアグラウンド・オブジェクト：全体が現在選択されているすべてのオブジェクトの前面にあるすべてのオブジェクト。３．ミッドグラウンドはバックグラウンドにもフォアグラウンドにもないすべてのコンポーネントからなっている。図３Ｂは、好適実施例によるグラフィック・モデルにおけるコンポーネントを示している。

この手法全体の主眼点は、トラック（ミッドグラウンド）期間に変化する最終イメージの部分を、変化しない部分から分離することである。カンバスは、DamageMidground、DamageForground およびDamageBackgroundなどの更新オブジェクト・メソッドをコールする。通常、アップデータは損傷を累積し、そのあとで必

要なものを必要になった時点で再ドローする。更新オブジェクトはこの情報を無視することを選択できることはもちろんである。更新オブジェクトの詳細説明は、種々のストラテジをどのようにインプリメントできるかを示している。カンバスはアップデータのDrawメソッドを必要なときにコールする。アップデータは更新する必要のあるバッファがあるかどうかを判断する。そのあと、アップデータはDrawBackground、DrawMidground、DrawForegroundを使用してコールバックしてカンバスに戻る。これらのルーチンは、カンバスが維持している（非公開：private）リスト全体にわたって繰り返して、該当のコンポーネントをドローイングするだけである。デフォルトのインプリメンテーションでは、DrawBackground内の他のすべてのコンポーネントの背後にあるグリッド（グリッドがオンの場合）をドローする。カンバスDrawXXXground メソッドは、アップデータ・オブジェクトからコールされることだけを目的としている。TGrafPort ポインタのほかに、これらのメソッドは再ドローイングする必要のあるエリアであるTGRectも取得する。最終的に、このパラーメータはTGAreaになることがある。これは、損傷したエリア(domaged area)の背後にあるすべてのコンポーネントのささいな拒否(trivial rejection)をサポートして、更新を高速化している。

カンバス・アクセサリ

カンバス更新

これらは、コンポーネント・カンバスの更新ストラテジを定義するオブジェクトである。これらは、カンバスのSetUpdaterメソッドを用いてカンバスにプラグインされる。バッファのないアップデータは損傷エリアを消去し、損傷エリアと交差するスクリーン上のすべてのオブジェクトを再ドローするだけで

ある。図４Ａは本発明による単一バッファ付きアップデータを示している。単一バッファ付きアップデータは単一の有効オフスクリーン・ビットマップ(valid offscreen bitmap) を維持している。これが再ドローのためにコールされたときは、スクリーンにだけブリット(blit)することができる。図４Ｂは、好適実施例による２重バッファ付きアップデータを示している。２重バッファ付きアップデータは２つのオフスクリーン・バッファを維持している。未選択のオブジェクト

とグリッドに似たバックグラウンドだけを収めているバックグラウンド・バッファと、修復オペレーションのときに使用されるコンポジット・バッファ(composit buffer)である。図４Ｃは、２重バッファ付き更新ポリシーを使用して再ドローするときのイベントのシーケンスを示している。カンバス・アップデータは、バッファなし更新、単一バッファ付き更新、２重バッファ付き更新、および３重バッファ付き更新に対するサポートを含んでいる。３重バッファリングは図４Ｄに示されている。３重バッファ付き更新は、全体が選択したオブジェクトの背後にあるすべての未選択オブジェクトを収めているバックグラウンド・バッファ４００、全体が選択したオブジェクトの前面にあるすべての未選択オブジェクトを収容しているフォアグラウンド・バッファ４２０、および修復オペレーションのとき使用されるコンポジット・バッファ４１０を維持している。すべてのバッファ付き更新ストラテジでは、メモリが消費される。カンバスが大きい場合には、カンバス全体をオフスクリーン・バッファにキャッシュすることは非合理的である（数メガバイトになる）。この場合は、スクリーンに現在表示しているカンバスの部分を収めているレクタングル（矩形）だけをキャッシュすることができる。バッファは、カンバスがサイズ変更またはズームされたときサイズ変更する必要があるが、カンバスがスクロールされるときも更新が必要である。

TCanvasUpdaterメソッドの説明

TCanvasUpdaterは抽象基底クラスであるので、そのコンストラクタは限定公開(protected)である。カンバスおよびグリッド、ページ、スナッパ(snapper)などの他のオブジェクトは、DamageBackground、DamageMidground およびDamageForegroundをコールすることができる。例えば、グリッドがバックグラウンドのすべてのコンポーネントの下にドローされる場合、そのグリッドが変化すると、DamageBackground をコールし、カンバスのエクステント(extent)を損傷エリアとして引き渡すことになる。これらのメソッドはディスプレイを即時に更新しない。その代わりに、カンバス・エリアの一部が有効でなくなったことをアップデータに通知する。サブクラスの有用なユティリティ・ルーチンとしてGetCanvas があり、このルーチンはアップデータが担当するカンバスを返却する。

GrafEditリリースには、TCanvasUpdaterのサブクラスとして、TUnbufferedUpdater、TSingleBufferedUpdater、TDoubleBufferedUpdaterの３つがある。これらは上述したストラテジ１～３に対応している。

ロケーション制約(constraining)フレームワーク

市販の大部分のドローイング・プログラムは、オプションの矩形グリッドをもっている。グリッドは、形状を作成または編集するときにドローイングを制約するために使用される。好適実施例によるグリッドの例をいくつか挙げると、次のとおりである。矩形、六角、等角（アーキテクチャによるドローイングでは、角度の傾きをもつ）、同心、半径、タイポグラフィ（アセンダ：ascender、ベースライン：baseline、ディセンダ：idescender）、チェスボード（８×８まで）、１点、２点または３点遠近グリッド、ページ・レイアウト・グリッドおよびシート・ミュジック用のスタッフ(staff)。どのグリッドもオン、オフすることができる。第２の独立変数は、グリッドを可視にするかどうかを制御するものである。グリッドには位置に影響されないものと、同心円グリッド・ラインをも

つ半径グリッドのように、位置に影響されるものとがある。後者のグリッド・タイプの場合は、グリッドに結合されるコンポーネントが用意され、直接操作を行うことができる。コンポーネントは、例えば半径グリッドの中心、または遠近グリッド(perspectine grid)の消尽(vanishing point)点を変更することができる。グリッドについては、カンバス・アクセサリの個所で詳しく説明する。デフォルトのグリッドはTRectangularGridのインスタンスである。

スナップツー(snap-to)・オブジェクト

好適実施例によれば、カーソルをグリッドへも、他のオブジェクト・フィーチャ（特徴）へもスナップ(snap)することが可能である。この機能は、正確な図面を迅速に作成するときに利用すると、極めて便利である。完全浮動点グラフィック・システム(fully floating point graphic system)では、図面がスケールアップとスケールダウンされて、解像度の異なるモニタから表示されるのが日常的であるので、オブジェクトへスナップ(snapping to object)することは不可欠である。その理由は、同じピクセルをヒットすることは、若干でもスケーリング、

移動または回転したあとでは、同じ座標が選択されたことを意味しないからである。図５はディスプレイ・スクリーン上の９個のピクセルを拡大して示した図である。四角の各々は１つのピクセルを表している。ポリラインは５１０に示されているが、これは指示したピクセルをクリックすると、ヘアラインとしてレンダリングされる。図面はポリラインを含めて別のドキュメントにコピーおよびペーストされて、若干スケーリングされる。５２０に示すように、ポリラインは見かけ上若干左へ移動している。最初のポリラインのコーナに結合するものとして、別のポリラインを追加するとき、コーナはまだ同じピクセル上に現れているが、説明するまでもなく、支援(help)がないと、シフトしたコーナ座標を正確にヒットすることが不可能になる。その代わりに、前述したようにピクセルの中心をクリックする。そこで、印刷時のように、図面をスケーリングまたは移動すると、スクリーンから見えていたものとドローされたものとの差が明確になる。スナップツー・オブジェクトは、他のオブジェクトの点とエッジの周囲に小

さい「重力」場を供給して、この問題が解決している。カーソルが重力場の近くにあるときは、カーソルは重力場の中心に「吸い込まれる」ことになる。スナッピング距離は半インチがデフォルトである。しかし、このスナッピング距離は最適なパフォーマンスが得られるように、ユーザが必要に応じて修正することができる。

セマンティック・スナッピング

非常にきれいで、正確な図面をドローイングすることのほかに、スナップツー・オブジェクトを他の場合に使用すると、セマンティック・フィードバックを得ることができる。図６は好適実施例によるスナップツー・オブジェクトを示している。コネクション６１０はTEcho ユニット６００からドラッグされている。コネクション６１０はグレーでドローされているが、これは、スピーカ６２０の一方のエンドポイントが浮いているためまだ未完成であることを示している。図７は、好適実施例によるセマンティック・スナッピング・オペレーションが完了したことを示している。図７に示すように、コネクションはスピーカ・ユニットの入力にスナップされている。ここで触れておきたいことがいくつかある。それは

カーソルがスピーカの入力ポートの「スナッピング半径」内で検出されると、「スナップ・エンタ・イベント(snap-enter event)」がトリガされていることである。タイプ・ネゴシーション（交渉）が行われ、アプリケーション固有の方法で処理されている。例えば、オーディオ・コネクションは、サウンド入力ポート上にドラッグすることができるが、ボリューム入力ポート上にドラッグすることはできない。第２に、カーソルがスナッピング半径内に留まっているかぎり、コネクションはスピーカ・ポートにスナップされたままになっていることである。これにより、ドラッグ期間になにかが実際に起こったとの重要なフィードバックが得られることになる。このフィードバックがないと、ある種のトラッカ(tracker)では、ユーザにポート上で「丁度で手を放す(just let go)」ことを要求する。それから、トラッカは、そのことのあとでヒット検出とタイプ・チェックを行う必要がある。その時点では、ミスを訂正するには遅すぎる。ス

ピーカ・ポート上で視覚的にロックすることのほかに、コネクションはブラックになり、マウスボタンを放すと正しいコネクションが得られることをユーザに知らせる。

一例として、コネクションがスピーカ・ユニットから離れるようにドラッグされたとする。カーソルがポートのスナップ領域から離れると、コネクションはアンロックし、エンドポイントは再びカーソルに追従していく。また、コネクションは再びグレーにドローされる。これは「スナップ・リービング・イベント(snap-leaving event)」と呼ばれている。アプリケーションは、スナップ・エンタ・イベントとスナップ・リービング・イベントのとき、オーディオ・フィードバックを与えることもできる。次に、コネクションがエコー・ユニットに戻るようにドラッグされたときは、フィードバックが起こり、エコー・ユニットがロックアップするので、コネクションは無効になる。スナップ・エンタ・イベントのときは、コネクションのエンドポイントは再び入力ポートにロックオンすることになる。タイプ・ネゴシエーションは正しいデータ・タイプ（型）を確認するが、トポロジ・チェックは円形経路を見つけ、コネクションが行えないことをユーザにフィードバックする。このフィードバックは、カーソルが入力ポートのスナッピ

ング半径内に残っているかぎり、その場所に留まっている。スナップ・リービング・イベントのときは、フィードバックは除かれるので、ユーザは自由に他の可能性を調べることができる。実際のトラック期間に広範囲なフィードバックを与えるようにすると、次のようなアプリケーションの利点が得られる。

－　正しくないデータ構造の作成を回避する。

－　「良好な」トラックの明示的確認を与える（つまり、手放すことはＯＫである）。

－　なにかが働かないときの理由をコンテキストにそった方法で説明する。

－　フォルトトラレントにすることにより反復的アクションを回避する（ツールを再びピックアップして、コネクション・ソースを再びターゲットにするといったように）。－いつでも手を引くことができる。

アイドルおよびトラッキング・スナップ・オブジェクト

コンポーネント・カンバスには、現在、アイドル・スナッパ(idle snapper)とトラッキング・スナッパ(tracking snapper)の２種類のスナップ・オブジェクトがある。これらのオブジェクトは共に同じタイプにすることができるが、通常は同じタイプではない。アイドル・スナッパが使用されるのは、それがオンにされた場合であり、マウスが作動しているときでもある。これは、アプリケーション・フレームワークのメイン・イベント・ループに実行すべきものがほかにないときコールされる。アイドル・スナップ・オブジェクトは、現在のロケーションでボタンが押されたとき、カーソルがそこにスナップするコンポーネントまたはロケーションを識別している。また、アイドル・スナップ・オブジェクトはビジュアル・フィードバックも提供する。割り当てられているアイドル・スナップ・オブジェクトは常に存在しているが、スナッピングはオフにすることができる。トラッキング・スナップ・オブジェクトは、トラッカがそのTrackContinue の中で使用するものである。トラッキング・スナッパはトラッカの種類が異なるごとに異なっている。例えば、TCreateRectTrackerは任意の他のコンポーネント上の任意の有意ロケーションにスナップすることができるが、TCreateConnectionTrackerはコネクション・ロケーションまたは正しいタイプと方向のコネクタにだけス

ナップする。アイドル・スナップ・オブジェクトとトラッキング・スナップ・オブジェクトのデフォルトはどちらも、TStandardSnap のインスタンスであり、スナッピングはオフにされている。

カンバス・グリッド(Canvas Grids)

グリッドはオブジェクトの制約されたドローイングと位置付けを容易にする。TCanvasGrid は抽象基底クラスであるので、すべてのコンストラクタは限定公開(protected)になっている。この基底クラスは、グリッド・スナッピングをオン、オフし、グリッド・ディスプレイをオン、オフするときのプロトコルを定義している。GetActive、SetActive、GetVisible、SetVisibleである。これらのメソッドはオブジェクト内部の論理型(Booleans)をセットするだけで、ドローイングも制約も行なわない。カンバスのDrawBackgroundメソッドは、グリッドのDrawメソッドをコールする。グリッドへのスナッピングの場合は、Constrain(制約)には２バージョンがある。最初のバージョンは１つのTGPoint 引数を受け取る。これはDoFirstTime メソッドの中でトラッカから呼び出される。Constrain のポイント・スナップ・バージョンは入力引数を最寄りのグリッド点(grid point)へスナップし、変更した点を返却する。これは、その点がすでに変更されていれば論理型も返却する。ある種のグリッドは、Consrainのこの最初のバージョンをまったく使用しないので、これらは常にFALSE（偽）を返却する。Constrain の２番目のバージョンは２つのTGPoint を受け取り、通常はトラッカのDoTrackContinue メソッドからコールされる。これは「方向性」スナップであり、若干多くなったコンテキストに基づいてスナッピングを利用する。例えば、ポリラインをドローイングするとき、水平ラインまたは垂直ラインだけにスナップすることもできる。等角(isonometric)および遠近グリッドもこの方向性スナップを使用する。Constrain のデフォルト・ポイント・スナップ・バージョンはなにも行わない。このデフォルト方向性スナップはその第２引数を使用して点スナップ・メソッドをコールする。これが合理的なデフォルトであるのは、グリッドが方向性スナップをもっていないとき、方向性スナップをオーバライドする必要がないからである。

MCollectibleルーチンと演算子は TCanvasGridに対しても実装されている。例えば、TRectangularGrid、TIsometricGrid、TPerspectiveGridのような、サンプル・グリッドがいくつか用意されている。図８は矩形グリッド(rectangulargrid）８００と等角グリッド８１０を示している。TRectangularGridは点の矩形グリッドにスナップする。これは方向性スナップをもっていない。等角グリッドは傾いた方向に沿って方向性スナップを追加する（角度はセット可能なパラメータである）。ポイント・スナップは斜めのグリッドに制約し、方向性スナップは水平、垂直または対角方向に制約する。この種のグリッドはアーキテクチャ・レンダリングやイラストレーションでよく使用されている。図９は好適実施例によるアーキテクチャ・レンダリングの例である。図９の大部分のラインは３つの主

要方向に追従しており、これは等角グリッドと同じである。TPerspectiveGridは２点遠近グリッドである。図１０は好適実施例による遠近グリッドを示している。水平ライン１０１０の高さと、水平ライン上の２つの消尽点のロケーションが指定されている。

遠近グリッド

２点遠近法では、描画されるすべてのラインは垂直であるか、消尽２点の一方に収束しているかのどちらかである。方向性スナップがこれを行う。ポイント・スナップは適用されない。図１１は、好適実施例に従い２点遠近グリッドを用いて作られた立方体の遠近描画を示している。

カンバス・スナップ

TCanvasSnap はスナップツー・オブジェクトの振舞いを定義するための抽象基底クラスである。他のオブジェクトへのスナッピングはグリッドへのスナッピングと似ている。そのため、TCanvasGrid と同じように、TCanvasSnap 基底クラスはオブジェクト・スナッピングをオン、オフし、スナップ・フィードバックをオン、オフするときのプロトコルを定義している。GetActive、SetActive、GetFeedbackVisible、SetFeedbackVisibleである。また、グリッドと同じように、これはポイント・スナップと、Constrain の方向性スナップ・バージョンを定義している。

スナップの遷移：SnapEnter、SnapLeave

スナップ・オブジェクトは、最後のスナップのタイプとロケーション、および最後のConstrain コールが実際にスナップを引き起こしたかどうかを記憶している。この情報はスナップ状態の変化を検出するために使用される。例えば、最後のConstrain コールはスナップしていなかったが、現在のコールがスナップしていれば、スナップ「領域」に入っている。メンバ変数を新しいスナップ・パラメータにセットしたあと、Constrain メソッドはSnapEnter をコールする。同様に、最後のConstrain コールはスナップを引き起こしていたが、現在のコールがスナップしていなければ、元のスナップ・パラメータを使用してSnapLeave をコールすることになる。Constrain への最後と現在のコールが共にスナップを引き起こしていたが、種類またはロケーションが異なっていれば、Constrain は元のスナップ・パラメータを使用してSnapLeave をコールし、新しいスナップを反映するようにメンバを更新したあとでSnapEnter をコールすることになる。Constrain への最後と現在のコールの間でなにも変化していなければ、コールは不要である。

スナップ・フィードバックをサポートするルーチンが他にもいくつかあるが、このスナップ・フィードバックは、現在のスナップのタイプとロケーションを示す、ある種のグラフィックまたはエクストラ・カーソルであるのが通常である。SnapEnter はグラフィックを表示し、SnapLeave はスナップ・フィードバックの下にある旧ビットを復元し、そのフィードバックを消去する。ルーチンDraw、（スナップ・フィードバック・グラフィックの）GetBounds およびIsSnapped はこのためのサポートを行う。通常、SnapEnter はDrawをコールし、SnapLeave はGetBounds をコールして修正すべきエリアを判別する。

グラフィック・エディタ・フレームワークは上述したクラスのほかに、複数の他のクラスから構成され、アプリケーション設計者がこれらのクラスを使用してグラフィック編集アプリケーションを作成できるメカニズムを提供している。開発者はデフォルトのクラスとアクションを使用することも、これらのデフォルトをオーバライドして、カストマイズしたクラスとアクションをインプリメントす

ることも可能である。

図１２は、好適実施例によるロケーション制約に関連する詳細ロジックを示すフローチャートである。処理は機能ブロック１２００から開始し、そこでマウスの動きまたは他のカーソルの相互作用(interaction)が検出されるまで待ち状態に入る。そのあと、機能ブロック１２１０で、トラック開始フェーズに入り、

カーソルの動きが検出されるとトラッキング・プロセスを開始する。トラッキング・プロセスは、トラッキング・プロセスが機能ブロック１２３０で終了するまで機能ブロック１２２０で続行される。

図１３は、好適実施例によるアイドル・スナップ・フェーズ（図１２の機能ブロック１２００）を示す詳細フローチャートである。処理は判定ブロック１３００から開始し、そこでアイドル・スナップ処理が現在アクティブであるかどうかを判定するテストが行われる。アイドル・スナップ処理がアクティブであれば、ロケーションが機能ブロック１３１０で制約され、ドロー・フィードバックが機能ブロック１３２０でユーザにフィードバックするためにアプリケーションに渡される。アイドル・スナップ処理がアクティブでなければ、バックグランド・グリッドがアクチィブであるかどうかを判定するテストが機能ブロック１３３０で行われる。グリッドがアクティブであれば、ロケーションが機能ブロック１３４０で制約され、コントロールが判定ブロック１３５０に渡される。判定ブロック１３５０で、マウスボタンが押されているかどうかを判定するテストが行われる。そうであれば、コントロールは機能ブロック１３６０に渡される。この機能ブロックは図１２の機能ブロック１２１０のロジックに対応するものである。マウスボタンが押されていなければ、コントロールは判定ブロック１３００に戻されて、アイドル・スナップがオンであるかどうかがテストされる。

図１４は、好適実施例によるトラック開始フェーズ（図１２の機能ブロック１２１０）に関連するロジックを示す詳細フローチャートである。処理は機能ブロック１４００から開始し、そこでコントロールがアイドル・フェーズから渡される（図１２の機能ブロック１２００の詳細である図１３）。次に、トラッキング・スナップ・オン処理がアクティブであるかどうかを判定するテストが判定ブロ

ック１４１０で行われる。そうであれば、ロケーションが制約され、機能ブロック１４３０で、ドロー・フィードバックがユーザに示されるためにアプリケーションに送り返される。トラッキングが判定ブロック１４１０でオンになっていなければ、判定ブロック１４８０で、バックグラウンド・グリッドがアクティブであるかどうかを判定するテストが行われる。そうであれば、機能ブロック１４９０でロケーションが制約される。そうでなければ、コントロールが機能ブロック１４４０に直接に渡され、ロケーションが処理される。次に、判定ブロック１４５０で、マウスボタンが押されたかどうかを判定するテストが行われる。そうであれば、コントロールは機能ブロック１４６０から図１５に詳細を示すトラック継続フェーズに渡される。そうでなければ、コントロールは機能ブロック１４７０から図１６に詳細を示すトラック終了フェーズに渡される。

図１５は、好適実施例によるトラック継続フェーズに関連するロジックを示す詳細フローチャートである。処理は機能ブロック１５００から開始し、そこでコントロールが図１４に詳細を示すトラック開始フェーズから渡される。マウスボタンが押されているかどうかを判定するテストが即時に判定ブロック１５１０で行われる。そうであれば、判定ブロック１５２０で、トラッキング・スナップがアクティブであるかどうかを判定するテストが行われる。そうであれば、機能ブロック１５３０で方向が制約され、機能ブロック１５４０で、ドロー・フィードバックがユーザに表示するためにアプリケーションに送り返される。マウスボタンが判定ブロック１５１０で押されていなければ、コントロールは機能ブロック１５６０から図１５に詳細を示すトラック終了フェーズに渡される。トラッキング・スナップが判定ブロック１５２０でアクティブでなければ、バックグラウンド・グリッドがアクティブであるかどうかを判定する別のテストが判定ブロック１５７０で行われる。そうであれば、ロケーションが機能ブロック１５８０で制約される。そうでなければ、コントロールが機能ブロック１５５０に渡されてロケーションが処理され、コントロールが判定ブロック１５１０に渡されてループが再び繰り返される。

図１６は、好適実施例によるトラック終了フェーズに関連するロジックを示す

詳細フローチャートである。処理は機能ブロック１６００から開始し、そこでコントロールがトラック開始（図１４）またはトラック継続（図１５）処理から渡される。トラッキング・スナップがアクティブであるかどうかを判定するテストが即時に判定ブロック１６１０で行われる。そうであれば、ロケーションが機能ブロック１６２０で制約され、ドロー情報をユーザに表示するために、フィードバックがアプリケーションへ送り返される。トラッキング・スナップが判定ブロック１６１０でアクティブでなければ、バックグラウンド・グリッドがアクティブであるかどうかを判定する別のテストが判定ブロック１６６０で行われる。グリッドがアクティブであれば、ロケーションが機能ブロック１６７０で制約され、コントロールが機能ブロック１６４０へ渡される。グリッドがアクティブでなければ、コントロールは機能ブロック１６４０へ渡され、そこでロケーションが処理され、コントロールが機能ブロック１６５０へ渡されて図１３に詳細を示すアイドル・フェーズに入る。

これまでに説明してきた関数に関連するC++コードを以下に示したのは、好適実施例をもっと分かりやすくするためである。

```
//- ©COPYRIGHT TALIGENT, Inc 1993
//    DoTrackFirstTime
//---------------------------------------
TTracker* TRectangleTracker::DoTrackFirstTime(TGPoint&p,const TEvent&)
{
   HandleSnapping(p);
   fRect=TGRect(p,p);
   return this;
}

//---------------------------------------
//   DoTrackContinue
//---------------------------------------
TTracker* TRectangleTracker::DoTrackContinue(TGPoint$p,const TEvent&)
{
   TGrafPort* onscreen=GetOnscreenDrawingPort();

   // 必要ならば方向を制限する
   HandleSnapping(GetOriginalClickLoc(),p);

   // カンバス更新をコールすることにより元のフィードバック下のビットを同期的に回復する
   // Invalidate()をここでビュー・システムにコールすることは、非同期であり、
   // 新しいrect(below) を描くと次に更新イベントにより上書きされる
   GetCanvas()->GetUpdater().Draw(fRect,onscreen);

   // 新しいマウス位置を反映するためにfRect を更新する
   fRect=TGRect(GetOriginalClickLoc(),p);

   // スクリーン上に直接新しいrectフィードバックを描く
   onscreen->Draw(fRect,TFrameBundle(TRGBColor(1.,0.,0.)));

   return this;
}

//---------------------------------------
//    DoTrackLastTime
//---------------------------------------
void TRectangleTracker::DoTrackLastTime(TGPoint&,const TEvent&)
{
   // カンバス更新をコールすることにより、元のフィードバック下のビットを回復する
   GetCanvas()->GetUpdater().Draw(fRect,GetOnscreenDrawingPort());
}

//---------------------------------------
//    DoDoneWithTracking
//---------------------------------------
void TRectangleTracker::DoDoneWithTracking()
{
   if(fRect.fLeft!=fRect.fRight||fRect.fTop!=fRect.fBottom){
       // fRect が退化していない場合、新しいrectコンポーネントを作成する
       TRectComponent* newRectComponet=newTRectComponent(fRect);
       newRectComponent->SetBundle(TGrafBundle(...));
       // TNewComponentCmd中にコンポーネントを包み、それを送出する
       TNewComponentCmdcmd(GetEncapsulator());
       cmd.AdoptComponent(newRectComponent);
       cmd.Do();
   }
   dekete this;
}
```

【図３】

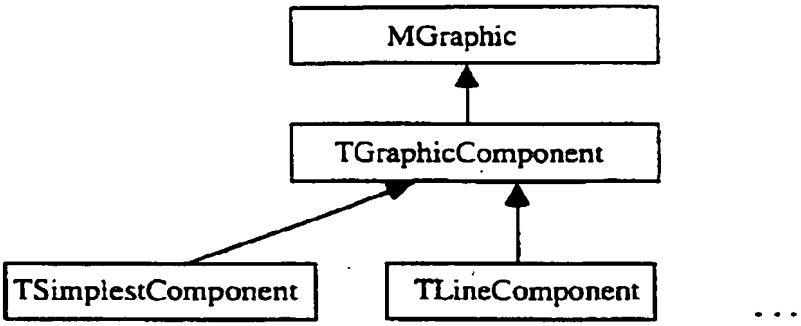


Figure 3A

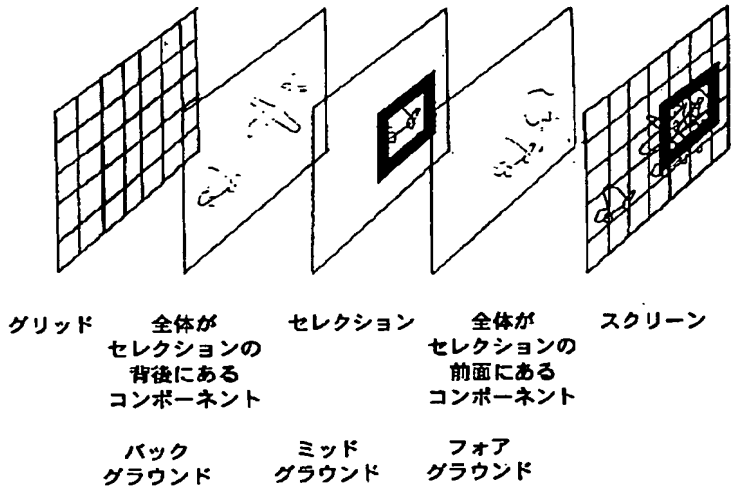


Figure 3B

【図４】

Figure 4A

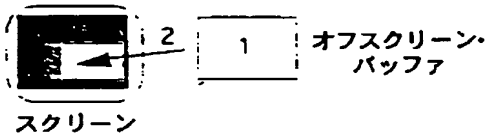


Figure 4B

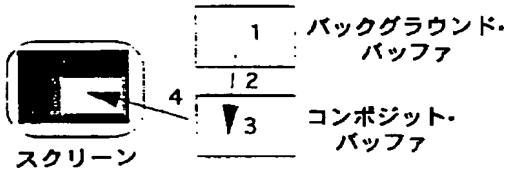


Figure 4C

【図４】

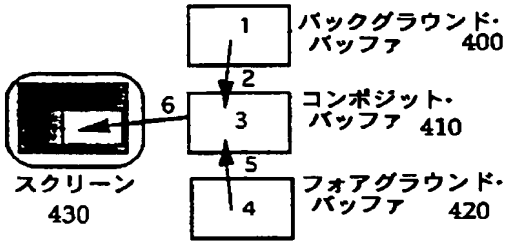


Figure 4D

【図５】

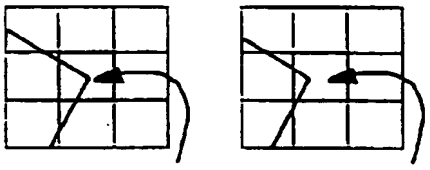


FIGURE 5

【図６】

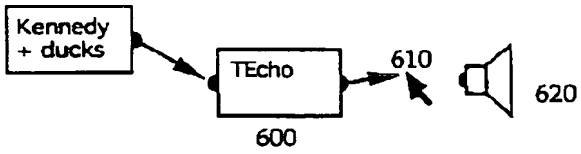


FIGURE 6

【図７】

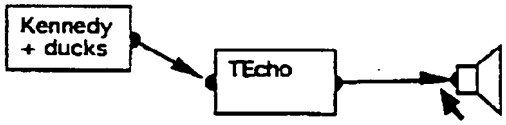
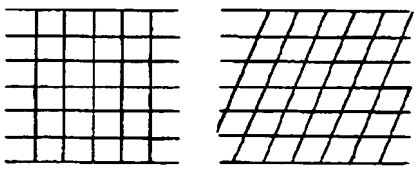


FIGURE 7

【図８】



800 810

FIGURE 8

【図９】

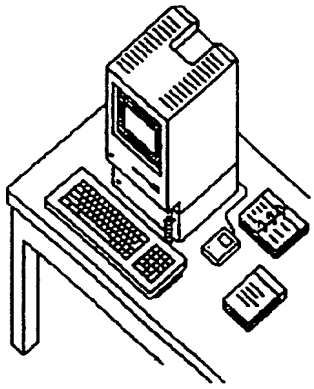


FIGURE 9

【図１０】

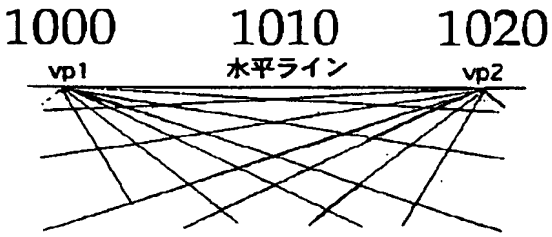


FIGURE 10

【図１１】

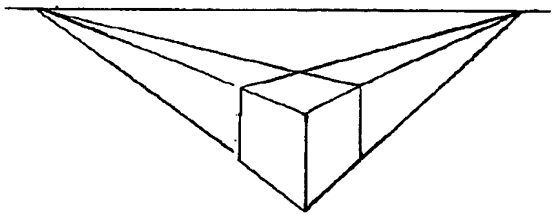


FIGURE 11

【図１２】

ロケーション制約フレームワーク･フローチャト

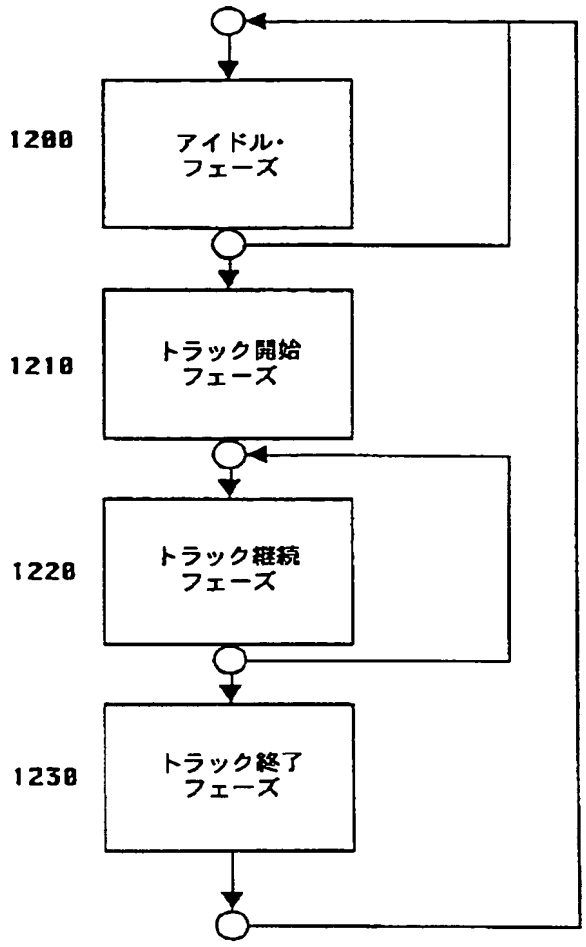


FIGURE 12

【図１３】

アイドル･スナップ･フェーズ

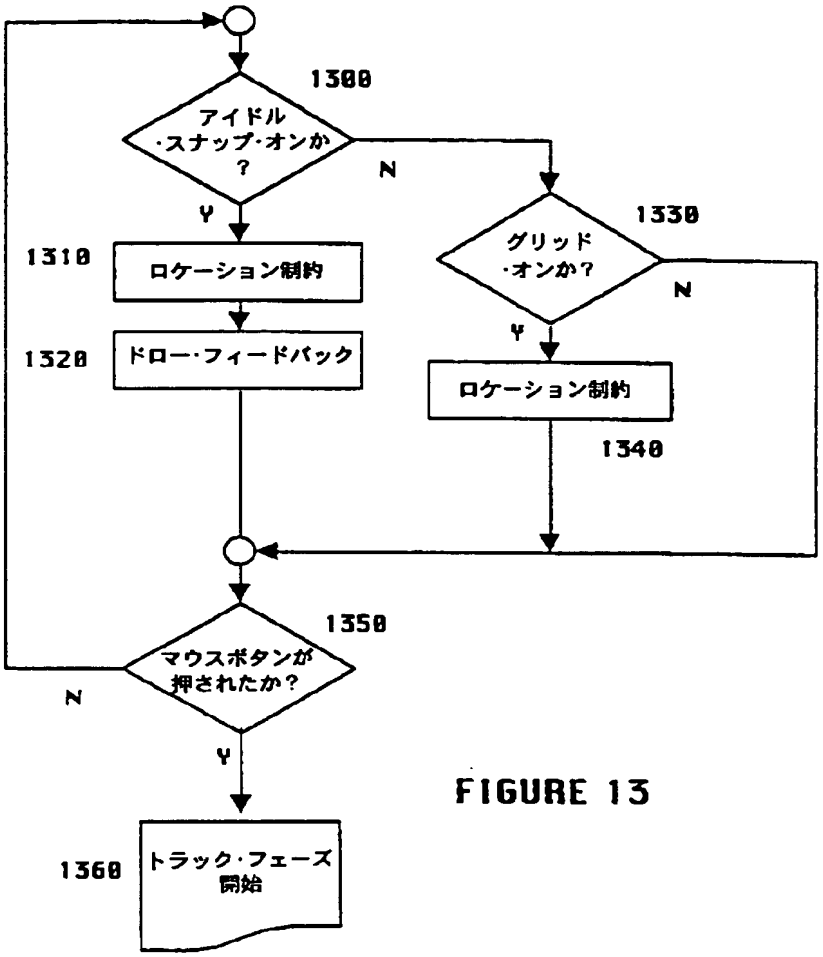


FIGURE 13

【図１４】

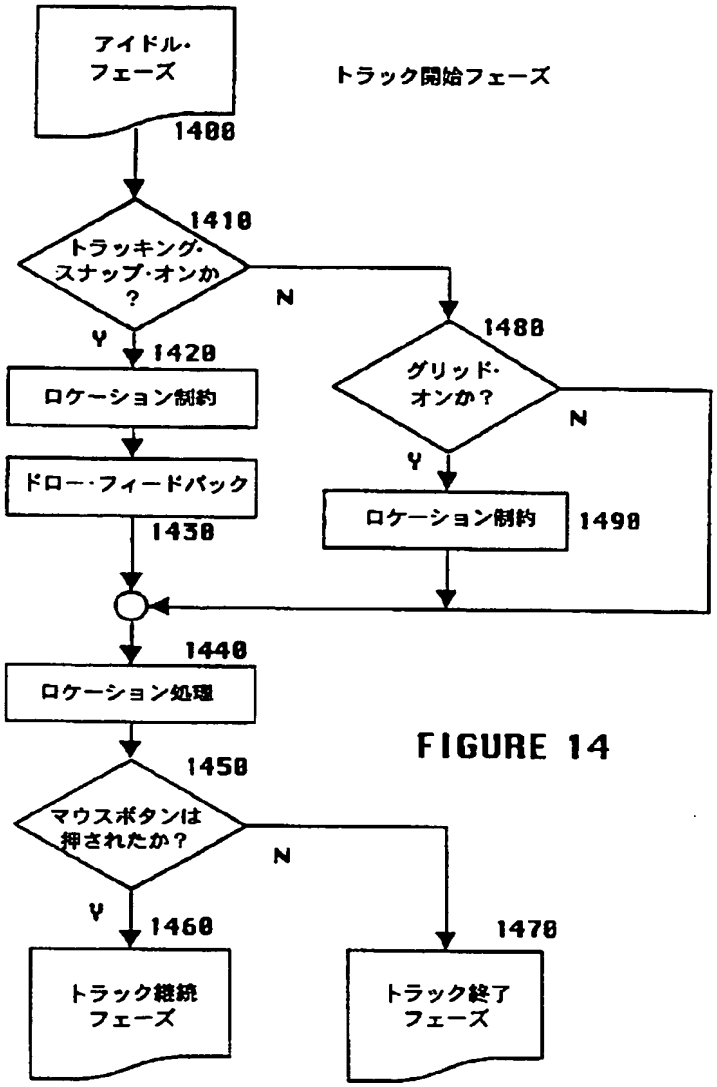


FIGURE 14

【図１５】

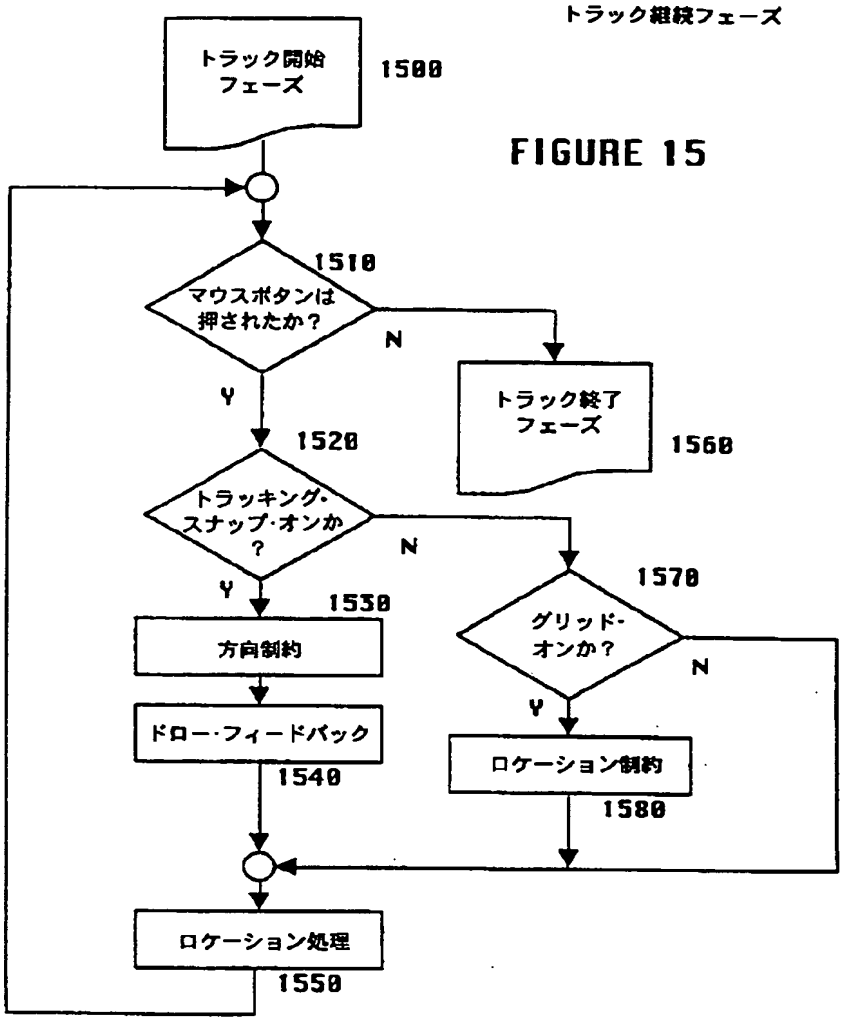


FIGURE 15

【図１６】

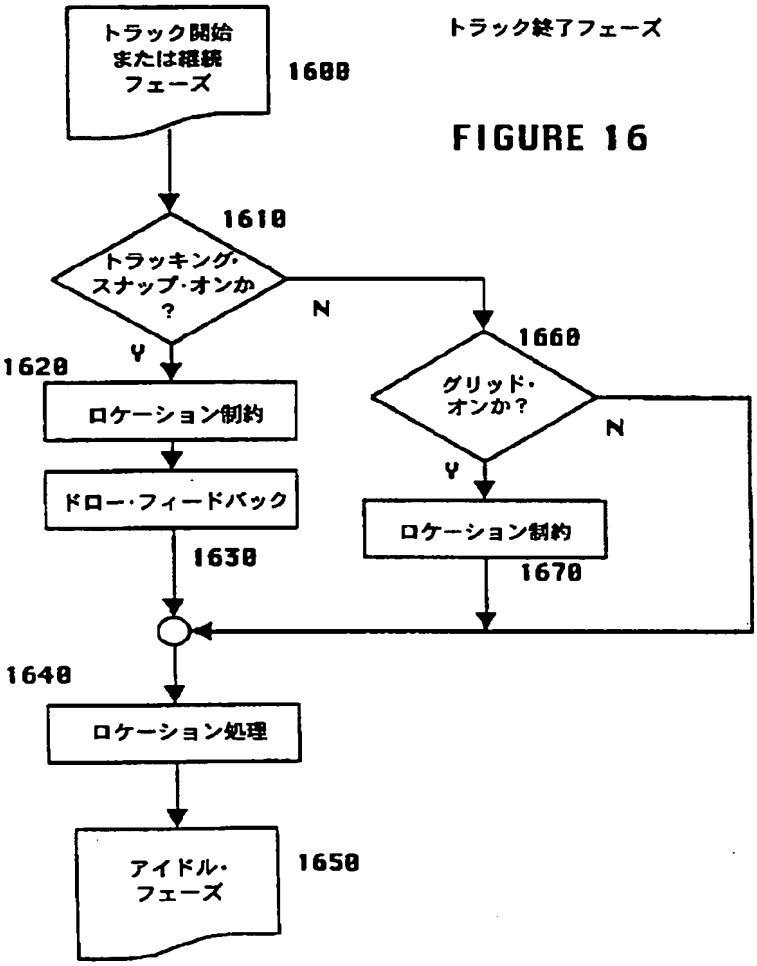


FIGURE 16

【手続補正書】特許法第１８４条の８

【提出日】１９９５年１０月２３日

【補正内容】

(原文明細書第２頁)

ＯＯＰの主題に関する詳細情報は、Grady Booch 著「オブジェクト指向設計とその応用(Object-oriented designs with Applications)」(Benjamin/Cummings Publishing Co., Inc., Redwood City, Calif.(1991)に記載されている。

Ｂｙｔｅ誌には、「Macapp: アプリケーション・フレームワーク」というタイトルの論文(August 1986, pp.189-193)の中でオブジェクト指向アプリケーション・フレームワーク・ツールが説明されている。この論文には、MacAppとそれをマッキントッシュ・システムで使用してアプリケーション開発を行う方法が説明されている。アプリケーションを開発するためにMacAppシステムで使用される基本コマンドのいくつかは、ドキュメントを作成する場合を中心にして説明されている。この論文では、グラフィック・エディタ・フレームワークは取り上げられていない。

Ｂｙｔｅ誌には、「パワーの分離(Separation of Powers)」というタイトルの論文(March 1989, pp. 255-262)の中でオブジェクト指向ユーザ・インタフェースが説明されている。この論文には対話式ソフトウェア・システムが説明されており、そこでは、アプリケーション層が仕事を行い、ユーザ・インタフェース層がやりとりを指示し、仮想（バーチャル）端末層が端末とのやりとりの詳細を取り扱っている。フレームワークはライブラリに似た構造を提供し、従来のプログラミング環境をサポートするために利用されている。MacAppアプリケーション・フレームワークは、TView クラスからサブクラス化されたウィンドウを作るための例として利用され、この例では、オープン、クローズ、サイズ変更といった標準的マッキントッシュ・ウィンドウの振舞いが説明されている。この論文では、グラフィック・エディタ・フレームワークは取り上げられていない。

発明の概要

従って、本発明の目的は、グラフィックス・アプリケーションを構築するためのオブジェクト指向フレームワークを提供することである。このフレームワーク

のオブジェクト指向フレームワークを提供することである。このフレームワークは、グラフィック情報をオペレーティング・システムのサブシステム相互間とアプリケーション内でやりとりすることを容易化するための、種々の関数を実現する多数のクラスを含んでいる。さらに、フレームワークは、より複雑な機能や関数を必要とする場合には、アプリケーション設計者がカストマイズしたり、オーバライドしたりできるようになっている。

本発明によれば、グラフィカル編集機能を使用してアプリケーションを開発するためのオブジェクト指向フレームワークが用意されており、このフレームワークには、システム・アーキテクチャのサブシステム相互間およびあるアプリケーションと他のアプリケーションとの間のデフォルト（省略時）のやりとりを定義するためのクラスが多数用意されている。クラスは、グラフィック・オブジェクトとデータをドローイング（描画）し、やりとりし、操作し、表示するためのメソッドを用意している。

図面の簡単な説明

上記および他の目的、側面および利点の理解を容易にするために、以下では、添付図面を参照して本発明の好適実施例について説明する。図面において、

図１は好適実施例によるコンピュータ・システムを示すブロック図である。

図２は、好適実施例においてグラフィカル・エディタ・フレームワークの基本的基底クラスが相互に作用し合う様子(interactio：やりとり)を示す図である。

図３Ａは、好適実施例においてあるコンポーネントの基底クラスと他のクラスとの関係を示す図である。

図３Ｂは、好適実施例によるグラフィック・モデルにおけるコンポーネントを示す図である。

図４Ａ、図４Ｂ、図４Ｃおよび図４Ｄは、好適実施例による種々の更新手法を示す図である。

図５は、好適実施例によるディスプレイ・スクリーン上の９個のピクセルを拡大して示す図である。

図６は好適実施例によるスナップツー(snap-to)・オブジェクトを示す図であ

る。

図７は、好適実施例によるセマンティック・スナッピング・オペレーションが完了したことを示す図である。

（原文明細書第９頁）

従って、基底クラスから派生するオブジェクトは基底クラスのインスタンスを表すために使用することができ、基底クラスがコールされるときは、いつでも、これを代用することができる。

グラフィカル・エディタ・フレームワーク(GrafEdit)には、基本的基底クラスとしてModel（モデル）、Component（コンポーネント）、Canvas（カンバス）の３つがある。図２は、これら３つの基本的基底クラス間の関係を示す概要図である。図２に示すように、Model は Componentを収容するのに対し、Canvasクラスは情報を表示し、Component のグラフィカル・オブジェクト・タイプに対する変更を反映している。

Model 基底クラスであるTGrafEditModelはデータ・ストア・クラスであり、複数のグラフィック・コンポーネントと他のモデル（GrafEditとそうでない場合の両方）を収容することができる。Model クラスはComponent メソッドへの基本アクセスを定義しているが、メソッドのストレージ・インプリメンテーションは定義していない。さらに、Model クラスはルート（根）Model として使用することができる。このクラスは、定義されたメソッドがコンポーネントと、従ってそのデータをアクセスするための、コンポーネントのストアの働きをする。デフォルトのインプリメンテーションでは、単純なフラット・ストレージ・システムである順序付リスト(ordered list)を使用して、コンポーネントをストアしている。異なるストレージ・システムが必要であるときは、開発者は、TGafEditModel からサブクラス化するという方法でこのシステムをオーバライドしてカストマイズすることができる。Model 基底クラスには、Model 内のコンポーネントに対してアクションを実行するメソッドがいくつか用意されている。

第１のメソッド群によると、Component データの追加、削除およびアクセスを行うことができる。別のメソッドとして、コンポーネントの順序を変更できるよ

うにするものが用意されている。これが重要であるのは、デフォルトとして、情報がスクリーン上にドローイングするために使用されるからである。Componentデータを走査するイタレータ(iterator)を作成するメソッドも用意されている。

イタレーション（繰返し）には、順序付(ordered)と、前面から後面へ(front to back)または順序なし(unordered)の２種類がある。Model 基底クラスには、Model 全体にわたるメニューを指定し、追加できるようにするメソッドも用意されている。最後に、フレームワークによって使用されるGrafEdit Selectionのすべてを作成するためのメソッドが用意されている。これにより、アプリケーション開発者はGrafEdit Selectionクラスをオーバライドすることができる。コンポーネントに影響するアクションを実行するために用意されたメソッドは、２通りの方法で実行することができる。最初は直接コールによる方法である。もう１つはModelCommandを使用してメソッドをコールする方法であり、このメカニズムによると、コマンドを取り消す(undo)ことが可能である。

請求の範囲

１．ディスプレイを装備するコンピュータ上で動作して、グラフィック変更コマンドに応答して複数のグラフィカル・イメージを編集し、表示するための対話式グラフィカル・ユーザ・インタフェースを構築するための装置であって、該装置は、

（ａ）複数のグラフィカル・イメージをグラフィック・コンポーネント・オブジェクトのリストとしてモデル化する手段であって、該グラフィカル・コンポーネント・オブジェクトの各々は、そこにストアされ、複数のグラフィック・イメージの１つを表わしているグラフィック・データと、該各々のグラフィック・コンポーネント・オブジェクトを選択する手段とを有し、

（ｂ）グラフィック・コンポーネント・オブジェクトのリストを収容すると共に、該リスト全体にわたって繰り返して、該複数のグラフィック・コンポーネント・オブジェクトの各々にストアされたグラフィック・データをディスプレイ上にレンダリングするモデル手段と、

（ｃ）前記モデル手段によって作成され、ディスプレイ上にレンダリングされたグラフィック・データを該モデル手段によって順序付けるためのカンバス・ビュー手段と、

（ｄ）該モデル手段に収容されていて、グラフィック変更コマンドに応答してグラフィック・コンポーネント・オブジェクトに収められたグラフィック・データを変更し、該変更されたデータをディスプレイからグラフィック・コンポーネント・オブジェクトに表示させる手段と

を備えたことを特徴とする装置。

２．モデル手段に収容されていて、ディスプレイの変更された部分を再レンダリングするためにスクリーン更新メカニズムを選択的にイネーブルする手段をさらに備えることを特徴とする請求項１に記載の装置。

３．グラフィック・データの変更された部分をディスプレイに直接にレンダリングするアップデータ・オブジェクトを含むことを特徴とする請求項２に記載の装置。

４．ディスプレイ全体のグラフィカル・イメージをストアするデータ構造とディスプレイ全体のグラフィカル・データをディスプレイにコピーするメンバ関数とをもつ単一バッファ付きアップデータ・オブジェクトを含むことを特徴とする請求項２に記載の装置。

５．２重バッファ付きアップデータ・オブジェクトを含み、該アップデータ・オブジェクトは、

未選択のグラフィック・コンポーネント・オブジェクトからのグラフィック・データを収めているバックグラウンド・バッファと、

ディスプレイ全体のグラフィカル・イメージの変更された部分を再ドローイングするためにディスプレイ全体のグラフィカル・イメージを収めているコンポジット・バッファと

を含むことを特徴とする請求項２に記載の装置。

６．未選択のグラフィック・コンポーネント・オブジェクトからのデータを収めているバックグラウンド・バッファと、選択されたグラフィック・コンポーネン

ト・オブジェクトからのグラフィック・コンポーネント・イメージ・データを収めているフォアグラウンド・バッファと、ディスプレイ全体のグラフィカル・イメージの変更された部分を再ドローイングするためにディスプレイ全体のグラフィカル・イメージを収めているコンポジット・バッファとを維持するための３重バッファ付きアップデータ・オブジェクトを含むことを特徴とする請求項２に記載の装置。

７．ディスプレイ上にグリッドを生成するグリッド・オブジェクトを含み、グリッドは複数のグリッド・ラインと複数のconstraint（制約）オブジェクトとからなり、該constraintオブジェクトの各々は、複数のグリッド・ラインの１つの

あらかじめ決めた半径内で選択された点を、該１つのグリッド・ラインへ移動するためのメンバ関数を収めていることを特徴とする請求項１に記載の装置。

８．複数のグラフィック・コンポーネント・オブジェクトの１つによって、ディスプレイ・スクリーン上にレンダリングされたグラフィカル・イメージを、ディスプレイ・スクリーン上の特定の位置に制約するconstraintオブジェクトを含むことを特徴とする請求項１に記載の装置。

９．複数のグラフィック・コンポーネント・オブジェクトの１つによって、ディスプレイ・スクリーン上にレンダリングされたグラフィカル・イメージを、あらかじめ決めた幾何学的パターンに沿って制約するconstraintオブジェクトを含むことを特徴とする請求項１に記載の装置。

１０．ドキュメント内の複数のグラフィカル・イメージを、ディスプレイ・スクリーンをもつコンピュータ・システム上で管理する方法であって、該方法は、

（ａ）複数のコンポーネント・オブジェクトを構築するステップであって、該複数のコンポーネント・オブジェクトの各々は、複数のグラフィカル・イメージの１つを定義しているデータと、該データに応答して該１つのグラフィカル・イメージをディスプレイ・スクリーン上にドローイングするメンバ関数とを有し、

（ｂ）複数のコンポーネント・オブジェクトの各々への参照をストアするためのデータ構造と、該複数のコンポーネント・オブジェクトへの参照全体にわたって繰り返し、モデル・プレゼンテーションを作成するメンバ関数とを有するモデ

ル・オブジェクトを構築するステップと、

（ｃ）ディスプレイ・スクリーン上に表示されるイメージのリストをストアするためのデータ構造を有し、表示されたコンポーネント・イメージが選択され、移動されたときカンバス・ビュー・オブジェクトに関連するビューを再ドローイングするカンバス・ビュー・オブジェクトを構築するステップと、

（ｄ）ドキュメント作成要求に応答して、カンバス・ビュー・オブジェクトを

構築するモデル・オブジェクト・プレゼンテーション作成メンバ関数をコールし、コンポーネント・オブジェクトの各々にあるドローイング・メンバ関数を順次コールするモデル・オブジェクト繰り返しメンバ関数をコールして、ディスプレイ・スクリーン上にグラフィカル・イメージをドローイングするステップと

を備えることを特徴とする方法。

１１．複数のコンポーネント・オブジェクトの各々は、各コンポーネント・オブジェクト内のデータを編集するメンバ関数と、各コンポーネント・オブジェクト内のデータを選択するメンバ関数とを含んでいて、該方法は、

（ｅ）複数のコンポーネント・オブジェクトの１つにある選択メンバ関数をコールして該複数のコンポーネント・オブジェクトの１つを選択するステップ

をさらに備えることを特徴とする請求項１０に記載の方法。

１２．モデル・オブジェクトは、コンポーネント・オブジェクトへの参照をデータ構造に追加し、コンポーネント・オブジェクトへの参照を削除するメンバ関数を含んでいて、ステップ（ｂ）は、

（ｂ１）追加メンバ関数をコールして、複数のコンポーネント・オブジェクトへの参照をモデル・オブジェクトに追加するステップ

を備えることを特徴とする請求項１０に記載の方法。

１３．カンバス・ビュー・オブジェクトはモデル・オブジェクトを識別するデータを含んでいて、ステップ（ｃ）は、

（ｃ１）ステップ（ｂ）で構築されたモデル・オブジェクトを識別するデータを用いてカンバス・ビュー・オブジェクトを構築するステップ

を備えることを特徴とする請求項１０に記載の方法。

１４．カンバス・オブジェクト・イメージ・リスト・データは、全体が現在選択されているすべてのオブジェクトの背後にあるバックグラウンド・オブジェクトのリストと、現在選択されているすべてのオブジェクトの前面にあるフォアグラウンド・オブジェクトのリストと、バックグラウンドにもフォアグラウンドにもないミッドグラウンド・オブジェクトのリストとを含んでいて、該方法は、

（ｃ２）コンポーネント・オブジェクトの各々をバックグラウンド・オブジェクトのリスト、フォアグラウンド・オブジェクトのリストおよびミッドグラウンド・オブジェクトのリストの１つに追加するステップ

を含むことを特徴とする請求項１０に記載の方法。

１５．さらに、

（ｆ）カンバス・オブジェクト・イメージ・リスト・データに応答して、表示されたコンポーネント・イメージが選択され、移動されたとき、カンバス・ビュー・オブジェクトに関連するビューを再ドローイングする更新オブジェクトを構築するステップと、

（ｇ）前記更新オブジェクトをカンバス・ビュー・オブジェクトに関連づけるステップと

を備えることを特徴とする請求項１０に記載の方法。

１６．さらに、

（ｈ）複数の更新オブジェクトを構築するステップであって、該更新オブジェクトの各々はカンバス・オブジェクト・イメージ・リスト・データに応答して、カンバス・ビュー・オブジェクトに関連するビューをあらかじめ定義した更新ストラテジに従って再ドローイングするステップ

を備えることを特徴とする請求項１０に記載の方法。

１７．さらに、

（ｉ）ディスプレイ・スクリーン上の複数のラインからなるグリッドを定義するためのデータと、該グリッドをディスプレイ・スクリーン上に表示するためのメンバ関数とを有するグリッド・オブジェクトを構築するステップと、

（ｊ）ステップ（ｉ）で構築されたグリッド・オブジェクトをカンバス・ビュ

一・オブジェクトに挿入するステップと

を備えることを特徴とする請求項１０に記載の方法。

１８．さらに、

（ｋ）表示された点を複数のグリッド・ラインの１つにスナップさせるメンバ関数を有するスナップツー・オブジェクトを構築するステップと、

（１）スナップツー・オブジェクトをカンバス・ビュー・オブジェクトに挿入するステップと

を備えることを特徴とする請求項１０に記載の方法。

【図１】

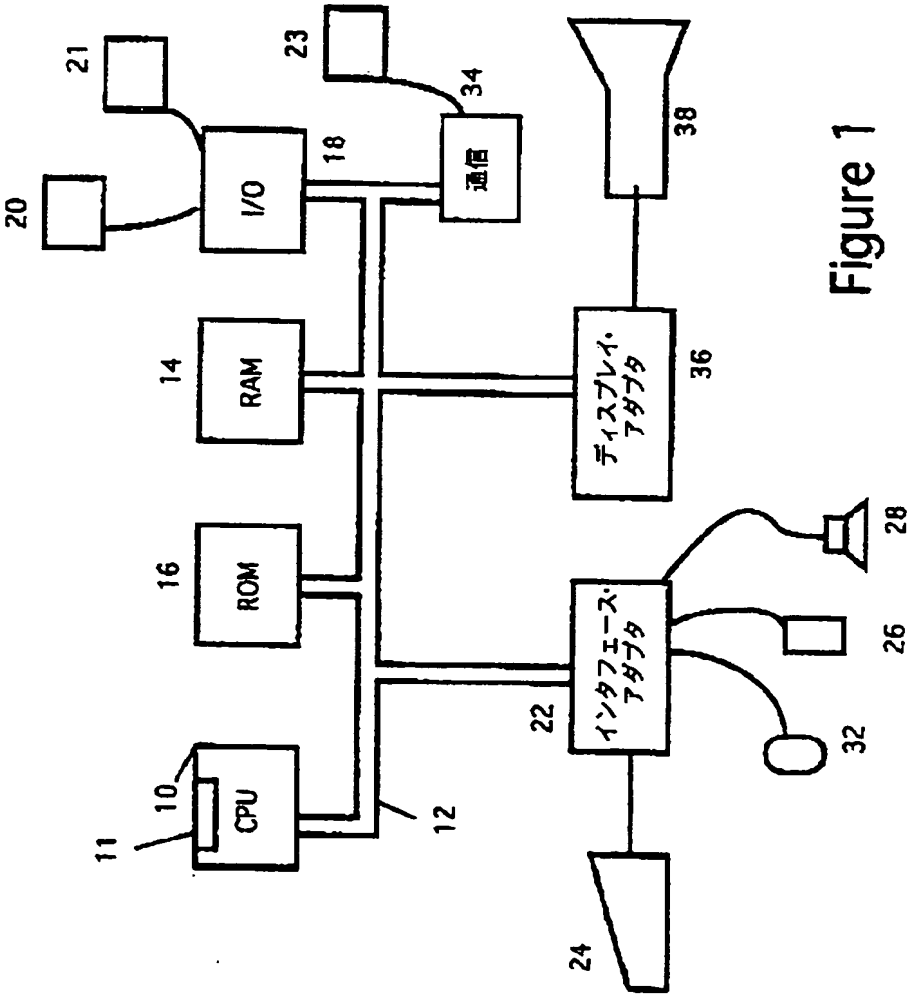


Figure 1

【図２】

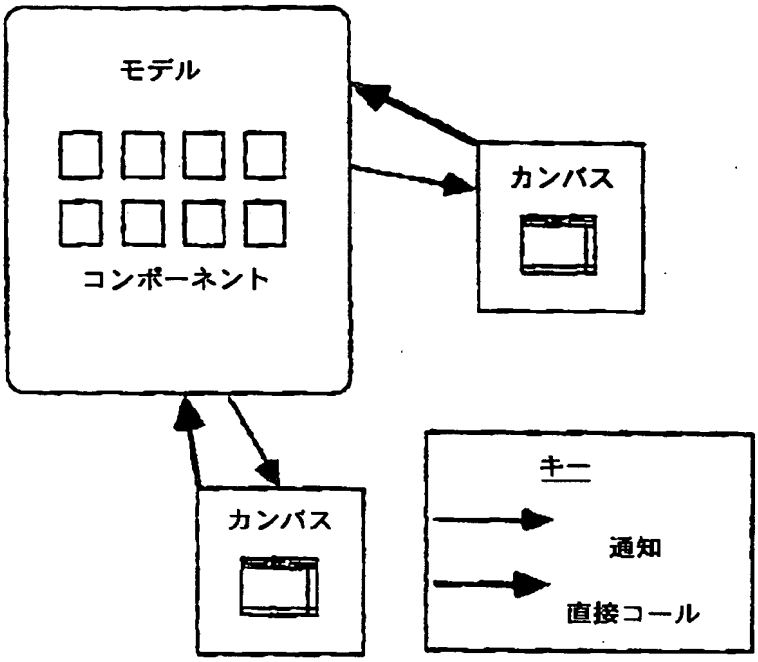


Figure 2

【国際調査報告】

INTERNATIONAL SEARCH REPORT

Intern al Application No
PCT/US 94/00054

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/44

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	BYTE. vol. 11, no. 8 , August 1986 , ST PETERBOROUGH US pages 189 - 193 SCHMUCKER 'MacApp: An application framework' cited in the application see page 189, left column, line 25 - middle column, line 15 see page 192, middle column, line 3 - page 193, middle column, line 15 --- -/--	1

[X] Further documents are listed in the continuation of box C. [ ] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier document but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
26 July 1994	29. 07. 94
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax (+31-70) 340-3016	Authorized officer Brandt, J

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Intern al Application No
PCT/US 94/00054

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	BYTE. vol. 14, no. 3 , March 1989 , ST PETERBOROUGH US pages 255 - 262 DODANI ET AL 'Separation of Powers' see page 256, right column, line 23 - page 258, left column, line 21 see page 261, middle column, line 3 - page 262, left column, line 4 -----	1,7-16, 19

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

フロントページの続き

(81)指定国 EP(AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OA(BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG), AT, AU, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, ES, FI, GB, HU, JP, KP, KR, KZ, LK, LU, LV, MG, MN, MW, NL, NO, NZ, PL, PT, RO, RU, SD, SE, SK, UA, UZ, VN

【公報種別】特許法第１７条第１項及び特許法第１７条の２の規定による補正の掲載
【部門区分】第６部門第３区分
【発行日】平成１３年６月１２日（２００１．６．１２）

【公表番号】特表平９－５０６１９１
【公表日】平成９年６月１７日（１９９７．６．１７）
【年通号数】
【出願番号】特願平７－５１２５７６
【国際特許分類第７版】
G06F 3/14 340
310
9/06 530
G06T 11/80
【ＦＩ】
G06F 3/14 340 A
310 C
9/06 530 N
15/62 322 B
320 K

手 続 補 正 書

平成１２年１２月２５日

特許庁長官 殿

１．事件の表示
特願平７－５１２５７６号
２．発明の名称
グラフィック・エディタ・フレームワーク・システム
３．補正をする者
オブジェクト テクノロジー ライセンシング コーポレイション
４．代 理 人
東京都港区赤坂２丁目６番２０号
電 話 (03)3589-4201（代表）
(7748) 弁理士 谷 穣 一
５．補正命令の日付
自 発
６．補正により増加する請求項の数 １７
７．補正対象書類名
明細書
８．補正対象項目名
請求の範囲
９．補正の内容
請求の範囲を別紙の通り補正する。

別 紙

請求の範囲

１． 複数のグラフィカルイメージを表示するためのメモリおよびディスプレイを備えるコンピュータシステム上で動作して、グラフィック変更コマンドに反応してイメージの変化によって引き起こされる障害を修復する装置であって、
該装置は、
（ａ）複数のグラフィカルイメージをメモリ内の複数のグラフィックコンポーネントオブジェクトとしてモデル化する手段であって、
該グラフィックコンポーネントオブジェクトの各々は、
複数のグラフィックイメージの１つを表わしてストアされたグラフィックデータと、ディスプレイ上で該各々のグラフィックコンポーネントオブジェクト中のグラフィックデータをレンダリングする手段とを有し、
（ｂ）前記複数のグラフィックコンポーネントオブジェクトの１つを選択し、該選択されたグラフィックコンポーネントオブジェクトを制御してディスプレイ上でグラフィックデータをレンダリングする手段と、ビュー損傷信号を生成するためのデータ変更コマンドに応答する手段とを含むカンバスビュー手段と、
（ｃ）前記カンバスビュー手段内に含まれ、前記ビュー損傷信号に応答して、予め決めたポリシーに従ってビュー損傷を修復する更新手段とを具えたことを特徴とする装置。

２． 前記更新手段は、
前記カンバスビュー手段により、グラフィックコンポーネントオブジェクトを選択し、該選択されたグラフィックデータをレンダリングする制御を行うことによって、
グラフィックデータの変更された部分をレンダリングする、非バッファ更新オブジェクトを含むことを特徴とする請求項１記載の装置。

３． 前記更新手段は、
ディスプレイ全体のグラフィカルイメージをストアするデータ構造と、ディス

プレイ全体のグラフィカルイメージをディスプレイにコピーするメンバ関数とを有する、単一バッファ更新オブジェクトを含むことを特徴とする請求項１記載の装置。

４．　前記更新手段は、
２重バッファ更新オブジェクトを含み、
ここで、該２重バッファ更新オブジェクトは、
未選択のグラフィックコンポーネントオブジェクトからのグラフィックデータを含むバックグラウンドバッファと、
ディスプレイ全体のグラフィカルイメージの変更された部分を再ドローイングするためにディスプレイ全体のグラフィカルイメージを含むコンポジットバッファと
を具えたことを特徴とする請求項１記載の装置。

５．　前記更新手段は、
３重バッファ更新オブジェクトを含み、
ここで、該３重バッファ更新オブジェクトは、
未選択のグラフィックコンポーネントオブジェクトからのデータを含むバックグラウンドバッファと、
選択されたグラフィックコンポーネントオブジェクトからのグラフィックイメージデータを含むフォアグラウンドバッファと、
ディスプレイ全体のグラフィカルイメージの変更された部分を再ドローイングするためにディスプレイ全体のグラフィカルイメージを含むコンポジットバッファと
を具えたことを特徴とする請求項１記載の装置。

６．　ディスプレイ上にグリッドを生成するグリッドオブジェクトを具え、
該グリッドは、複数のグリッドラインと、複数のｃｏｎｓｔｒａｉｎｔ（制約）オブジェクトとからなり、
該ｃｏｎｓｔｒａｉｎｔオブジェクトの各々は、前記複数のグリッドラインの１つの予め決められた半径内で選択された点を、１つのグリッドラインへ移動するメンバ関数を含むことを特徴とする請求項１記載の装置。

７．　前記複数のｃｏｎｓｔｒａｉｎｔオブジェクトの少なくとも１つは、
前記複数のグラフィックコンポーネントオブジェクトの１つによってディスプレイスクリーン上にレンダリングされたグラフィカルイメージを、ディスプレイスクリーン上の特定の位置に制約することを特徴とする請求項６記載の装置。

８．　ディスプレイスクリーン上のグラフィックイメージの部分に基づいてグラフィックイメージの外観を変更するために、セマンチックｃｏｎｓｔｒａｉｎ：を使用するｃｏｎｓｔｒａｉｎ：オブジェクトを含むことを特徴とする請求項７記載の装置。

９．　前記ｃｏｎｓｔｒａｉｎ：オブジェクトは、
ディスプレイ上でレンダリングされたグラフィカルイメージのあらかじめ決められた部分を制約し、
該グラフィカルイメージのあらかじめ決めた部分は、グラフィカルイメージの角、縁、中央および中心点の最初の１つに一致することを特徴とする請求項８記載の装置。

１０．　前記ｃｏｎｓｔｒａｉｎｔオブジェクトは、
ディスプレイ上でレンダリングされたグラフィカルイメージに一致するグラフィカルコンポーネントの結合部分を、ディスプレイ上のあらかじめ決められた部分に整合させることを特徴とする請求項８記載の装置。

１１．　各グラフィックコンポーネントオブジェクトは、
ユニークな識別値をディスプレイ上でレンダリングされた複数の選択されたグラフィカルイメージに与える手段と、
複数のグラフィカルコンポーネントの中から選択されたグラフィカルコンポーネントの間での比較において使用するために、データストアオブジェクトの各々のユニークな識別値をストアする手段と
を含む請求項６記載の装置。

１２．　ドキュメント内の複数のグラフィカルイメージをメモリおよびディスプレイスクリーンを有するコンピュータシステム上で管理する装置であって、
該装置は、前記グラフィカルイメージに対する変更によって引き起こされるビュー損傷を修復し、
（ａ）メモリ内の複数のコンポーネントオブジェクトと、
ここで、該複数のコンポーネントオブジェクトの各々は、複数のグラフィカルイメージの１つを定義しているデータと、該データに応答して該１つのグラフィカルイメージをディスプレイスクリーン上にドローイングするメンバ関数とを有し、
（ｂ）複数のコンポーネントオブジェクトの各々への参照をストアするためのデータ構造を有する、メモリ内のモデルオブジェクトと、
（ｃ）メモリ内のカンバスビューオブジェクトを構築する手段と、
ここで、該カンバスビューオブジェクトは、ディスプレイスクリーン上に表示されるイメージを有するグラフィックコンポーネントオブジェクトのリストをストアするためのデータ構造と、表示されたコンポーネントイメージが選択されて移動されたとき、ビュー損傷を示す方法とを有し、
（ｄ）複数の更新オブジェクトと、
ここで、該複数の更新オブジェクトの各々は、あらかじめ決められたポリシーに従ってカンバスビューオブジェクトに関連するビューを再ドローイングする方法を有し、
（ｅ）１つの更新オブジェクトが、あらかじめ決めた方法で損傷を受けたエリアを再ドローイングするために示されたビュー損傷に応答するように、前記更新オブジェクトの１つを前記カンバスビューオブジェクトに関連づける手段と
を具えたことを特徴とする装置。

１３．　前記複数のコンポーネントオブジェクトの各々は、
各コンポーネントオブジェクト内のデータを編集するメンバ関数と、各コンポーネントオブジェクト内のデータを選択するメンバ関数とを含むことを特徴とする請求項１２記載の装置。

１４．　前記モデルオブジェクトは、
前記コンポーネントオブジェクトへの参照を前記データ構造に追加し、前記コンポーネントオブジェクトへの参照を削除するメンバ関数を含むことを特徴とする請求項１２記載の装置。

１５．　前記カンバスビューオブジェクトは、モデルオブジェクトを識別するデータを含むことを特徴とする請求項１２記載の装置。

１６．　前記カンバスオブジェクトイメージリストデータは、
全体が現在選択されている全てのオブジェクトの背後にあるバックグラウンドオブジェクトのリストと、
現在選択されている全てのオブジェクトの前面にあるフォアグラウンドオブジェクトのリストと、
前記バックグラウンドのリストにも前記フォアグラウンドのリストにもないミッドグラウンドオブジェクトのリストと
を含むことを特徴とする請求項１２記載の装置。

１７．　前記カンバスビューオブジェクトは、グリッドオブジェクトを含み、
該グリッドオブジェクトは、
ディスプレイスクリーン上の複数のラインからなるグリッドを定義するためのデータと、
該グリッドをディスプレイスクリーン上に表示するためのメンバ関数と
を有することを特徴とする請求項１２記載の装置。

16. 前記カンバスビューオブジェクトは、
表示された点を複数のグリッドラインの1つにスナップさせるメンバ関数をする、スナップツーオブジェクトを含むことを特徴とする請求項17記載の装置。

19. ドキュメント内の複数のグラフィカルイメージを、メモリおよびディスプレイスクリーンを有するコンピュータシステム上で管理する方法であって、
(a) 前記メモリ内の複数のコンポーネントオブジェクトを構築するステップと、
ここで、該複数のコンポーネントオブジェクトの各々は、
複数のグラフィカルイメージの1つを定義しているデータと、
該データに応答して該1つのグラフィカルイメージをディスプレイスクリーン上にドローイングするメンバ関数とを有し、
(b) 前記メモリ内のモデルオブジェクトを構築するステップと、
ここで、該モデルオブジェクトは、複数のコンポーネントオブジェクトの各々への参照をストアするためのデータ構造を有し、
(c) 前記メモリ内のカンバスビューオブジェクトを構築するステップと、
ここで、該カンバスビューオブジェクトは、
ディスプレイスクリーン上に表示されるイメージを有するグラフィックコンポーネントオブジェクトのリストをストアするためのデータ構造と、
表示されたコンポーネントイメージが選択されて移動されたとき、ビュー損傷を示す方法とを有し、
(d) 前記カンバスビューオブジェクト中の更新オブジェクトを構築するステップと、
ここで、該更新オブジェクトは、あらかじめ決められたポリシーに従ってビューダメージを修復する方法を有し、
(e) あらかじめ決められた方法で示されたビュー損傷を修復するために、前記更新オブジェクトの修復方法がビュー損傷信号に応答してコールされるように、前記更新オブジェクトを前記カンバスビューオブジェクトに関連づけるステップ
と
を具えたことを特徴とする方法。

20. 前記複数のコンポーネントオブジェクトの各々は、
各コンポーネントオブジェクト内のデータを操作するメンバ関数と、
各コンポーネントオブジェクト内のデータを選択するメンバ関数とを含み、
該方法は、
(f) 該複数のコンポーネントオブジェクトの1つにある選択メンバ関数をコールして、該複数のコンポーネントオブジェクトの1つを選択するステップをさらに具えたことを特徴とする請求項19記載の方法。

21. 前記モデルオブジェクトは、
コンポーネントオブジェクトへの参照を前記データ構造に追加し、コンポーネントオブジェクトへの参照を削除するメンバ関数を含み、
前記ステップ(b)は、
(b1) 追加メンバ関数をコールし、複数のコンポーネントオブジェクトへの参照をモデルオブジェクトに追加するステップを具えたことを特徴とする請求項19記載の方法。

22. 前記カンバスビューオブジェクトは、モデルオブジェクトを識別するデータを含み、
前記ステップ(c)は、
(c1) 前記ステップ(b)で構築されたモデルオブジェクトを識別するデータを用いて、カンバスビューオブジェクトを構築するステップを具えたことを特徴とする請求項19記載の方法。

23. 前記カンバスオブジェクトイメージリストデータは、
全体が現在選択されている全てのオブジェクトの前後にあるバックグラウンドオブジェクトのリストと、
現在選択されている全てのオブジェクトの前面にあるフォアグラウンドオブジェクトのリストと、
前記バックグラウンドのリストにも前記フォアグラウンドのリストにもないミッドグラウンドオブジェクトのリストを含み、
該方法は、
(c2) 前記コンポーネントオブジェクトの各々を、前記バックグラウンドオブジェクトのリスト、前記フォアグラウンドオブジェクトのリスト、前記ミッドグラウンドオブジェクトのリストの1つに追加するステップ
を含むことを特徴とする請求項19記載の方法。

24. (g) 前記メモリ内の複数の更新オブジェクトを構築するステップと、
ここで、該複数の更新オブジェクトの各々は、あらかじめ決められた更新ストラテジーに従ってカンバスビューオブジェクトに関連するビューを再ドローイングするために、カンバスオブジェクトイメージリストデータに応答し、
(h) 複数の更新オブジェクトの1つを、カンバスビューオブジェクトに関連づけるステップと
をさらに具えたことを特徴とする請求項19記載の方法。

25. (i) ディスプレイスクリーン上の複数のラインからなるグリッドを定義するためのデータと、該グリッドをディスプレイスクリーン上に表示するためのメンバ関数とを有するグリッドオブジェクトを構築するステップと、
(j) 前記ステップ(i)で構築されたグリッドオブジェクトをカンバスビューオブジェクトに挿入するステップと
をさらに具えたことを特徴とする請求項19記載の方法。

26. (k) 表示された点を複数のグリッドラインの1つにスナップさせるメンバ関数を有するスナップツーオブジェクトを構築するステップと、
(l) 前記スナップツーオブジェクトをカンバスビューオブジェクトに挿入するステップ
をさらに具えたことを特徴とする請求項25記載の方法。

27. 前記ステップ(d)は、
(d1) ディスプレイ全体のグラフィカルイメージをストアするデータ構造と、
ディスプレイ全体のグラフィカルイメージをディスプレイにコピーするメンバ関数とを有する単一バッファ付き更新オブジェクトを構築するステップ
を含むことを特徴とする請求項19記載の方法。

28. 前記ステップ(d)は、
(d2) 未選択のグラフィックコンポーネントオブジェクトからのグラフィックデータを含むバックグラウンドバッファと、
ディスプレイ全体のグラフィカルイメージの変更された部分を再ドローイングするために、ディスプレイ全体のグラフィカルイメージを収めているコンポジットバッファと
を具えた2重バッファ更新オブジェクトを構築するステップ
を含むことを特徴とする請求項19記載の方法。

29. 前記ステップ(d)は、
(d3) 未選択のグラフィックコンポーネントオブジェクトからのデータを含むバックグラウンドバッファと、
選択されたグラフィックコンポーネントオブジェクトからのグラフィックイメージデータを収めているフォアグラウンドバッファと、
ディスプレイ全体のグラフィカルイメージの変更された部分を再ドローイングするためにディスプレイ全体のグラフィカルイメージを収めているコンポジットバッファと
を保持するための3重バッファ付き更新オブジェクトを構築するステップ
を含むことを特徴とする請求項19記載の方法。

30. （1）前記複数のグラフィックコンポーネントオブジェクトの1つによって、前記ディスプレイ上にレンダリングされたグラフィカルイメージを、該ディスプレイ上の特定の位置に制約するスナップをさらに具えたことを特徴とする請求項19記載の方法。

31. ドキュメント内の複数のグラフィカルイメージを、メモリおよびディスプレイスクリーンを有するコンピュータシステム上で管理するコンピュータプログラム製品であって、

該プログラム製品は、コンピュータが読み取り可能なプログラムコードを有するコンピュータの使用できる媒体を具え、

（a）前記メモリ内の複数のコンポーネントオブジェクトを構築するプログラムコードと、

ここで、該複数のコンポーネントオブジェクトの各々は、

複数のグラフィカルイメージの1つを定義しているデータと、

該データに応答して該1つのグラフィカルイメージをディスプレイスクリーン上にドローイングするメンバ関数とを有し、

（b）前記メモリ内のモデルオブジェクトを構築するプログラムコードと、

ここで、該モデルオブジェクトは、該複数のコンポーネントオブジェクトの各々への参照をストアするためのデータ構造を有し、

（c）前記メモリ内のカンバスビューオブジェクトを構築するプログラムコードと、

ここで、該カンバスビューオブジェクトは、

前記ディスプレイスクリーン上に表示されるイメージを有するグラフィックコンポーネントオブジェクトのリストをストアするためのデータ構造と、

表示されたコンポーネントイメージが選択されて移動されたとき、ビュー損傷を示す方法を有し、

（d）メモリ内の更新オブジェクトを生成するプログラムコードと

ここで、該更新オブジェクトは、あらかじめ決められたポリシーに従ってビュー損傷を修復するために、カンバスビューオブジェクトと関連するビューを再ドローイングする方法を有することを特徴とするコンピュータプログラム製品。

32. 前記複数のコンポーネントオブジェクトの各々は、

各コンポーネントオブジェクト内のデータを編集するメンバ関数と、

各コンポーネントオブジェクト内のデータを選択するメンバ関数とを含み、

ここで、該コンピュータプログラムは、

（f）前記複数のコンポーネントオブジェクトの1つにある選択メンバ関数をコールし、該複数のコンポーネントオブジェクトの1つを選択するプログラムコードをさらに含むことを特徴とする請求項31記載のコンピュータプログラム製品。

33. （g）前記メモリ内の複数の更新オブジェクトを構築するプログラムコードをさらに具え、

ここで、該複数の更新オブジェクトの各々は、

他の更新オブジェクトによって用いられる更新ストラテジーと異なるあらかじめ決められた更新ストラテジーに従ってカンバスビューオブジェクトに関連するビューを再ドローイングするために、カンバスオブジェクトイメージリストデータに応答することを特徴とする請求項31記載のコンピュータプログラム製品。

34. （h）ディスプレイスクリーン上の複数のラインからなるグリッドを定義するためのデータと、該グリッドをディスプレイスクリーン上に表示するためのメンバ関数とを有する、メモリ内のグリッドオブジェクトを構築するプログラムコードと、

（i）前記グリッドオブジェクトを前記カンバスビューオブジェクトに挿入するプログラムコードと

をさらに含むことを特徴とする請求項31記載のコンピュータプログラム製品。

35. （j）表示された点を複数のグリッドラインの1つにスナップさせるメンバ関数を有するスナップツーオブジェクトを構築するプログラムコードと、

（k）前記スナップツーオブジェクトを前記カンバスビューオブジェクトに挿入するプログラムコードと

をさらに含むことを特徴とする請求項34記載のコンピュータプログラム製品。

（以下余白）